Institut für Pflanzenbau und Pflanzenzüchtung II
Justus-Liebig-Universität Gießen
Professur für Biometrie und Populationsgenetik
Prof. Dr. Matthias Frisch

PopSim: A Parallelized Simulation Software for Population Genetics

Dissertation zur Erlangung des akademischen Grades eines
Doktor der Naturwissenschaften
- Dr. rer. nat. -

an der Justus-Liebig-Universität Gießen

Vorgelegt von
Arsh Rup Singh
aus Amritsar, Indien

Gießen, 08 Juni 2012

Die vorliegende Arbeit wurde am 08 Juni 2012 als “Dissertation zur Erlangung des Grades einses Doktors der Naturwissenschaften (Dr. rer. nat.)” angenommen.

Tag der mündlichen Prüfung	06 August 2012
Berichterstatter, 1. Prüfer:	Prof. Dr. Matthias Frisch
Mitberichterstatter, 2. Prüfer:	Prof. Dr. Thomas Wilke
3. Prüfer:	PD Dr. Rod Snowdon
4. Prüfer:	Prof. Dr. Bernd Honermeier

Contents

1. Introduction

Finding conclusive solutions for problems in population genetics involves very time-consuming and tedious procedures and furthermore, many times the available analytical solutions are impractical or there is no well-defined analytical solution available at all. To overcome such complex problems, computer simulation has come to the aid as a very powerful and useful tool. By using computer simulations, one can solve highly complicated problems, which are non-practical or are too time-intensive to be solved analytically. Though the computer simulations make it possible to solve these problems, they have some inherent problems of their own. One potential problem is that computer simulations are very heavy on system resources. The computer simulation programs that are available today were designed for computers containing a single processor. However, during the past few years, computers with multiple processors and/or computers using processors containing multiple cores have become the norm. In order to fully utilize the processing power of these new computers, it is required and is essential that the simulation software should be parallelized.

Population Genetics is the science of investigating how the genetic constitution of a given population changes with time under the influence of natural selection, genetic drift, mutations, and gene flow. It provides the theoretical framework necessary for studying the effects of crossing different genetic strains over a given time period. A tractable mathematical model where options for selection, finite population size, and planned mating is not possible with analytical solutions.

Simulation software is a class of computer programs that are used to observe the effect of a process, operation or experiment using a set of mathematical formulas without actually performing the process or operation physically. The benefit of using the simulation software are immense, both in saving time and costs. “The Population Genetics Simulation Software” simulates the effect of random or planned crossing between different sets of parents on the genetic variation in the population. Achieving some conclusive results on a simple population study involving a group of individuals, that in reality would take years to finish, can be obtained within minutes if not seconds using simulation software.

The downside of having a computer with multiple cores is that older programs designed for sequential execution on a single core are not able to utilize the gain in performance provided by the multiple cores. For a program to be actually able to utilize the benefit of the multiple cores, it should be designed in such a way that it could run multiple instances of itself or its sub-parts at the same time. The software has to be designed in such a way that it breaks the problem into discrete parts, which can then be executed concurrently by different cores of a computer system. Software designed to work in this manner is known as “parallelized software”.

Whenever any software is designed to solve a problem by utilizing the multiple cores of the system and running parts of the routines simultaneously, a completely new set of obstacles like; race conditions, mutual exclusion, synchronization and parallel slowdown have to be tackled. To deal with these impediments in the case of population genetics simulations requires finding and creating new scientific concepts on data structure and algorithms.

By parallelizing the “Simulation Software for Population Genetics”, we can enhance the speed of execution of the program greatly, depending on the system configuration, thus reducing the time a researcher needs to wait for the simulation to finish processing. However, to do this the new software has to surmount various hindrances posed by the concurrent execution of the various parts of the simulation. Since there is no currently available population genetics simulation software or model that is able to run the various parts of the simulation in parallel, new concepts for the simultaneous operation of various parts of the simulation need to be devised. Furthermore, newer algorithms designed specifically to handle population genetics problems need to be created.

1.1 Existing Software

There are a number of software programs available for performing simulations on population genetics, but most of these are designed to perform only a specific task and do not provide the functionality required for many other types of studies. Moreover, these are not designed to utilize the multi-core architecture of modern computers. The most important ones are:

GENEPOP (Version 1.2) is a population genetics software package for exact tests and ecumenicism for haploid or diploid data *(Raymond and Rousset 1995)*. It mainly performs two tasks: (i) It computes exact tests or their unbiased estimation for Hardy-Weinberg equilibrium, population differentiation, and two-locus genotypic disequilibrium. (ii) It converts the input *GENEPOP* file to formats used by other popular programs, like BIOSYS (*Swofford and Selander 1981*), LINKDOS (*Garnier-Gere and Dillmann 1992*), and Slatkin's (*1993*) isolation-by-distance program, thereby allowing communication between them.

QU-GENE is a flexible and powerful platform for investigation of the characteristics of genetic systems undergoing repeated cycles of selection and mating. The core of the system is the E (N:K) genetic model, where E is the number of types of environment, N is the number of genes, K indicates the level of epistasis and the parentheses indicate that different N:K genetic models can be nested within types of environments. It uses a two-stage architecture that separates the definition of the genetic model and genotype-environment system from the detail of the individual simulation experiments. There are typically three steps involved in running a *QU-GENE* simulation – (i) specification of the genetic-environment system, (ii) specification of a mating and selection scheme and (III) running the simulation *(Podlich and Cooper 1998).*

Easypop (Version 1.7) is a computer program for population genetic simulations that simulates both haploid and diploid organisms (*Balloux 2001*). It allows the proportion of cloned and sexual reproduction to be selected in case of haploid organisms. In case of diploid organisms with one sex, it allows the proportion of selfing to be selected and in case of bisexual diploid organisms, it gives the choice between hermaphrodites and sexual organisms. It has been implemented in various models for migrations such as two-dimensional stepping-stone and hierarchical island model. *Easypop V1.7* can handle a maximum population size of 10,000 individuals with a maximum of 999 allelic states.

SelSim is a program to simulate population genetic data with natural selection and recombination written in C++ for Monte Carlo simulation of DNA polymorphism data, for a recombining region within which a single bi-allelic site has experienced natural selection. It

provides a number of different mutation models for simulating surrounding neutral variation. Within a coalescent framework, *SelSim* allows the simulation from either a fully stochastic model or deterministic approximations to natural selection *(Spencer and Coop 2004)*.

SIMCOAL 2.0 is a program to simulate genomic diversity over large recombining regions in a subdivided population with a complex history, performs simulation of the genomic diversity, of samples drawn from a set of populations with arbitrary patterns of migrations and complex demographic histories, including bottlenecks and various modes of demographic expansion *(Laval and Excoffier 2004)*.

Arlequin V3 is an integrated software package for population genetics data analysis and is a Windows only software package written in C++, integrating several basic and advanced methods for population genetics data analysis *(Excoffier, Laval and Schneider 2005)*. Some of the functionalities offered by *Arlequin V3* are; computation of standard genetic diversity indices, estimation of allele and haplotype frequencies, departure from linkage equilibrium tests, departure from selective neutrality and demographic equilibrium tests, etc.

simuPOP is a forward-time population genetics simulation environment that can simulate large and complex evolutionary processes *(Peng and Kimmel 2005)*. It is based on a scripting language (Python) that provides a large number of objects and functions to manipulate populations and a mechanism to evolve populations forward in time. It allows users to create, manipulate and evolve populations interactively and in batch mode by using a script provided by the user. A number of built-in scripts are provided in *simuPOP*, which can perform simulations ranging from implementation of basic population genetics models to generating datasets under complex evolutionary scenarios.

TreesimJ is a flexible, forward time population genetic simulator that can perform simulations on sampling of genealogies, genetic data and many population parameters from populations evolving under complex evolutionary scenarios *(O'Fallon 2010)*. Many fitness and demographic models are provided in the application along with the option to create

custom models. It assumes that each individual has exactly one parent, thus diploidy, recombination and sexual selection are not considered and treated.

1.2 Base Work

Simulation software *Plabsim* (*Frisch et al. 2000*) and *Plabsoft* (*Maurer et al. 2008*) were developed in our working group and these programs have been used in more than 50 refereed publications. The comparison of simulated and experimental datasets showed a very good fit of the simulation models used in these programs (*Prigge et al. 2008*).

Plabsim is a computer program for simulating marker-assisted selection in arbitrarily designed backcross programs. *Plabsim* can evaluate the simulated data for gene frequency, genotype frequency, frequency of homozygous loci, length of chromosome segments originating from one ancestor and the number of marker data points required for a breeding program. In addition to data analysis, *Plabsim* can also export the simulated data for analysis with statistical software.

Plabsoft is powerful and flexible simulation software capable of performing population genetics simulations and data analysis. It can simulate various mating systems comprising planned crosses, random mating, selfing, partial selfing, single-seed descent, double haploids, top-crosses and factorials. It provides data analysis routines for analyzing simulated and experimental datasets for allele and genotype frequencies, genotypic and phenotypic values and variances, molecular genetic diversity, linkage disequilibrium and parameters to optimize marker-assisted backcrossing programs.

1.3 Objectives

The objectives of the present project were to study various approaches for parallelization, find the most suitable approach for implementation and also to explore and study the working of already existing simulation software *Plabsoft* and *Plabsim;* and finally to design new parallelized simulation software *PopSim* based on *Plabsoft*.

The main goals for the new simulation software were that the new program should be able to utilize the advantage provided by the multiple core architecture of modern

computers, which until now no other software in this category is utilizing. In addition, it should be completely backwards compatible with *Plabsoft* commands i.e. scripts written for *Plabsoft* should be able to run on *PopSim* without the need to make any changes in the script apart from the name of the library to load. An additional prerequisite was that unlike the older programs the new program should not be dependent on any third party libraries like GSL, GMP, etc. Further it was also necessary that the new program should be able to compile both on Linux and Windows based machines with no or minimal differences between the code for the two operating systems. Any weaknesses that are found in *Plabsoft* should also be removed and the algorithms suitable for parallelization should be detected. The new software was required to compile cleanly without giving any errors or warnings on both the Linux and Windows System and it should also be able to compile on a compiler not supporting parallelization. Lastly, the new software was expected to be able to run faster than *Plabsoft* at least for the parallelized part, while keeping the results similar and comparable to the *Plabsoft* results.

The *PopSim* software that has to be developed would be an add-on package for the "*R S*tatistical Software" *(Ihaka and Gentleman 1996)*. It would be a flexible and powerful parallelized simulation software for population genetics and data analysis capable of fully utilizing the multi-core architecture of today's computers. It would be able to handle a broad range of problems concerning plant breeding and genetics. *PopSim* would be able to simulate the various mating scenarios like random mating, selfing, partial selfing, planned crosses, double haploids, top-crosses, single-seed descent and factorials. It would provide means to simulate selections according to selection indices based on molecular marker scores and/or phenotypic values. A range of data analysis routines would be provided in *PopSim* to analyze simulated and experimental datasets for allele and genotype frequencies, genotypic and phenotypic values and variances, molecular genetic diversity, linkage disequilibrium and parameters to optimize marker-assisted backcrossing programs using simulated and experimental datasets.

2. Methods

A new Population Genetics Simulation software based on *Plabsoft* was to be designed to handle various problems of Population Genetics, It was required that it should provide different types of validated and performance optimized functions. These functions could also be used by the user to create new high-level user-defined functions in combination with the data analysis routines provided by R and other add-on packages of the R system called in arbitrary order. The entire list of the functions provided in the new Population Genetics Simulation software known as *PopSim* along with their intended purpose or functionality has been provided in *Appendix I*.

Earlier *Karlin and Liberman (1978*) had simulated meiosis by the *Count-Location Process*, this is the very same process used by *PopSim* for the simulation of meiosis. This *Count-Location Process* is a two-step process whereby realization of Poisson-distributed random variable 'M' is determined by using parameter 'λ' in the first step (count). The second step determined the locations of 'k' crossovers with realizations of a uniformly distributed random variable. This algorithm makes two assumptions. First it assumes that the average number of crossovers formed on a given chromosome are equal to the length of the chromosome in Morgan units. Second, it assumes that all the locations of the crossovers are uniformly distributed on the chromosome. These assumptions imply the absence of interference (*Stam 1979*) and are mathematically equivalent to those underlying mapping function of *Haldane (1919)*.

The $G = \sum_{S \subseteq N} X_s$ equation was employed to model the genotypic value 'G' of an individual for a certain trait. In this case 'N' was the set of all loci on both homologous chromosomes and 'X_S' was an effect assigned to a given combination of alleles at a subset of loci S (*Bulmer 1985*). This model permitted a flexible definition of genetic effects, including additive, dominance, and epistatic effects of any order. Moreover, an arbitrary number of traits could be simulated in this model, each with its own genetic architecture. Phenotypic values were simulated by adding non-genetic effects to the genotypic values according to arbitrary field designs or error structures.

In addition, *PopSim* like *Plabsoft* does not distinguish between experimental and simulated data sets. In *PopSim* linkage map, marker and trait data from experimental studies can be easily imported from text files, and the base populations for simulations could be described by specifying relevant population genetic parameters. Once the dataset has been transferred to *PopSim* by one of the above-mentioned methods, the provided dataset could be directly analyzed by using data analysis routines, or the dataset could be used to conduct simulations, which could be subsequently followed by data analysis.

In the present case, *PopSim* has been designed to be an add-on package for the '*R* statistical software'. The algorithms that were computationally demanding and the data management routines were coded in the '*C* programming language' *(Kernighan and Ritchie 1988)* while R was used for the coding of less demanding routines/functions. '*OpenMP*' *(Architecture Review Board 1997)* application-programming interface (API) was used to parallelize the C language part of the code and it was compiled with '*GCC*' *(Stallman and Tower 1987)*. *PopSim* has been designed to be able to run under both *Microsoft Windows* and the *Linux* operating systems. The Linux platform does not require any add-ons to be installed while the Windows' implementation has the additional requirement of *Pthreads-w32 (Elliston and Johnson 1998)*, which adds the support for *POSIX Threads (IEEE Std 1003.1c-1995)* that are natively supported by Linux. All these methods, which have been used in the creation of *PopSim*, are described in detail below:

R is a statistical computing and graphics language and environment containing software facilities for data manipulation, calculation and graphical display. It consists of an effective data handling and storage facility and provides a suite of operators that can be used to perform array calculations, especially for matrices. For data analysis, it provides a large coherent and integrated collection of intermediately tools. The graphical facilities available for data analysis give the option either of displaying the results on on-screen or on a hardcopy. Along with all these features R is a well developed, simple and effective programming language that includes input and output facilities along with conditional operators, loops, user-defined recursive functions, and everything else that a programming language of the ALGOL-type provides.

Over time, R has become one of the most used statistical languages and most statisticians consider it to be the de facto standard for developing statistical software. R can be regarded as an open source implementation of the language *'S' (Becker and Chambers 1981)* and even though differences exist between the two, code written for S can normally be run under R with little alteration.

Similar to S, R is also designed around a true computer language and it permits users to add additional functionalities by defining new functions. To expand the capabilities of R, it allows the users to link C, C++ and FORTRAN code for computationally intensive tasks, which are then called at runtime. It even gives the advance users the option to manipulate the R objects directly from the C code. Compared to other statistical languages R has much stronger object-oriented programming facilities. Furthermore, due to R's permissive lexical scoping rules it is easy to extend R, therefore users can employ packages made by other users for specific functions or areas of study. Because of its ability to use static graphics, R can show very high quality graphs as well as mathematical symbols. Further capabilities to handle dynamic and interactive graphs can be also added to R by installing extra packages.

Matrices are inherently implemented in R and thus it is possible to do matrices calculations like addition, inversion, etc. without using any loops. Other data structures included in R are scalars, vectors, matrices, data frames and lists.

There are two types of functions in R: *functions* and *generic functions*. The generic functions behave according to the type of argument that is passed to them, i.e. they recognize the type of object and select the method accordingly, and are object oriented whereas the normal functions only support procedural programming.

R was originally created at the University of Auckland, New Zealand by Ross Ihaka and Robert Gentleman. The name R was conceptualized partly after the first letter of the first names of the two authors programming and partly because the letter 'R' comes just before the letter 'S' (denoting the S language from which R is inspired).

Since R is part of the GNU project, its source code is freely available under the GNU General Public License along with pre-compiled binary for various operating systems. The default interface of R is a command line interface but there are several third party graphical user interfaces available for R.

C, an imperative language, is one of the most popular programming languages of all time and its compilers are available for almost all of the different types of computer architectures, from embedded-microcontrollers to supercomputers. It was originally designed by Dennis Ritchie at Bell Telephone Laboratories for implementing system software in the *UNIX (Thompson, Ritchie, Kernighan, McIlroy and Ossanna 1969)* operating system. Even though the C language was designed to implement system software, it has been widely used for developing application software.

The C language was designed to provide low-level access to computer memory with its constructs mapping efficiently to machine language and in the process requiring minimal run-time support. Another design feature of C was that it compiled using a relatively straightforward compiler thus adding the possibility to code applications, which were previously coded in assembly language.

One of the main design strengths of the C language is that it was designed with cross-platform programming in mind. Due to this, a program which is compliant to C standards and written portably is able to compile on a very wide variety of computer architectures and operating systems with just a few changes in the source code.

Due to C's low runtime demand on system resources, ability to access specific hardware addresses and match externally imposed data access requirements, it is often used to implement operating systems and embedded system applications. Using CGI as a "gateway" between the browser, server and the web application, C can be used for website programming as well.

Many compilers, libraries and interpreters of other programming languages are often implemented in C. In addition, programs that perform a lot of computations are coded in C

because it allows efficient implementations of algorithms and data structures due to its thin layer of abstraction and low overhead.

Some implementations of other languages use C as an intermediate language, which gives the other languages the convenience of code portability as it is no longer dependent of the machine-specific code generators. C is not ideal for use as an intermediate language as it was originally designed as a programming language, not as a compiler target language.

The main characteristics exhibited by C language are; the entire executable code in C is contained within functions and the parameters of various functions are always passed by value. The pass-by-reference of function is simulated in C by explicitly passing pointers (address of the memory location). C gives the option to combine and manipulate related data elements as a unit by using *struct*. The concept of recursion is fully supported in the C language. Being a free-format language C uses the semicolon as a statement terminator. In C, functions and data fully support run-time polymorphism. Low-level access to computer memory is provided by converting machine addresses to typed pointers. Complex functionality such as mathematical functions, input/output and string manipulation are provided by library routines. The C language provides a large number of compound operators like +=, -=, *=, ++, etc. while only a small set of keywords are reserved. It allows the programmer to hide the variables in nested blocks. In C, characters can be used as integers because it uses partially weak typing. It provides a preprocessor for macro definition, inclusion of source code from header files and conditional compilation. The array indexing is defined in terms of pointer arithmetic. It uses logical "and" "or" operators which are represented by && and ||, respectively. The right operand is not evaluated if the result can be determined from the left alone. These logical operators are semantically distinct from the bit-wise operators & and |. Also the '=' is used for assignment while '==' is used to test for equality. The C language delimits the compound statement blocks by {…}, which are also used to define blocks for nesting.

Although C has a formal grammar specified by the C standard, the source code is free-form, which allows arbitrary use of whitespace to format the code, rather than column-based or text-line-based restrictions.

A very important aspect of a programming language is its facilities of memory management and management of objects stored in memory. There are three different types of memory management routines provided by C: Automatic - objects are stored in a stack, and this space is freed automatically after the block, which contains the object, is exited. Static - space for the object is provided in the binary at compile-time and the memory space is reserved for the entire time the binary (which contains the object) is in memory. Dynamic - blocks of arbitrary size are requested at run-time using library functions like malloc, the memory is reserved until it is freed by the user by calling the appropriate function.

Libraries are used in C language as a primary method of extension. A library consists of an archive file containing all the functions, as well as a header file containing the prototypes of the various program-accessible functions and declarations of special data types and macro symbols used with these functions. Numerous libraries are available for C, which add various functionalities to the C language. These libraries are often written in C because the C compilers generate efficient object code. Programmers have even created interfaces to these libraries so that higher-level languages can use the routines in the libraries.

C was widely used to implement end-user applications, but due to the advent of newer and easier to learn languages, much of that development has now shifted to the newer languages.

OpenMP (Open Multi-Processing) is an application-programming interface for C, C++ and FORTAN, which supports multi-platform shared memory multiprocessing on Linux, UNIX, Microsoft Windows and other platforms. It is a scalable and portable model for programmers, consisting of a set of compiler directives, library routines and environment variables that influence run-time behavior and provides very simple and flexible interface for developing parallelized applications for different types of platforms ranging from the desktop computer to the supercomputers.

OpenMP implements parallelization by using multithreading, where a task is divided into a master thread which forks into a specified number of slave threads at places where

parallelization is possible. The runtime environment allocates these slave threads to different processors thus running them concurrently. After the execution of the parallelized code, all the threads join back into the master thread which then continues. If any thread is executed before the others it can be made to wait for the rest of the slave threads to complete before the code is executed further or the code can be moved further while some of the threads are still executing depending upon the algorithm and the requirements of the program.

A preprocessor directive *(#pragma omp parallel*) is used to mark the section of the code that is to be parallelized. This directive will inform the compiler, which causes the threads to be formed before the execution of this section. All of the created treads are assigned a unique id which can be obtained by calling the function *omp_get_thread_num( )*. The thread id of the master tread is integer value '0' and the ids of the remaining threads have different integer values.

In C/C++, OpenMP uses #pragmas to tell the compiler where the OpenMP commands are coded in the source code. OpenMP uses the *omp parallel* command to fork out the code enclosed by the construct into a separate thread, which will be run parallel to the main program thread. The original process will be denoted as master thread with thread id 0.

Although the different threads execute the parallelized part of the code independent of each other by default, it is possible to divide a task among the threads using "work-sharing constructs" so that each thread executes the section of the code allocated to it. This way it is possible to achieve both task parallelism and data parallelism in OpenMP.

The various work-sharing constructs used to specify how to assign independent work to one or all of the threads are: (i) *omp for or omp do*, also known as loop constructs, these constructs are used to split the loop iterations into different threads. (ii) *sections*, these are used to assigning consecutive but independent code blocks to different threads. (iii) *single*, this construct specifying a block of code which is to be executed by only one thread, a barrier is implied in the end. (iv) *master*, this construct is very similar to single, but the code block will be executed by the master thread only and no barrier implied in the end.

The number of threads can be assigned by the runtime environment based on environment variables or it can be coded in the source code using functions.

With OpenMP, it is theoretically possible to get a speedup factor of N while running a parallelized program on a system with N processors compared to a system with only one processor, but this is not the case normally because some part of the program may not be parallelized, as some algorithms require sequential execution. In addition, the N processors may have the N times computational power but the memory bandwidth is not of the scale of N and normally the memory path is shared by the different processors. Along with these the common problems affecting other parallelization methods like load balancing and synchronization overhead are applicable on OpenMP.

These days OpenMP is implemented by most of the major commercial and non-commercial compilers.

GCC (GNU Compiler Collection) produced by the GNU Project www.gnu.org is a collection of compilers for different programming languages. The GCC is widely deployed as a compiler tool in closed source, proprietary and commercial software development environments, thus it has been ported to a wide variety of processor architectures ranging from supercomputers to desktop computers to embedded systems and even the modern day videogame consoles.

The external interface of the GCC is standard like any other UNIX compiler. A driver program named *gcc* is invoked by the user and arguments are passed to this program. These arguments are interpreted by the *gcc* and a decision is made on which compiler to use for the input file. The *gcc* program then runs the assembler and the linker, if required, to produce a complete executable binary.

There is a separate compiler program for each supported language, whose input is the source code of that particular language and it outputs the assembly code. All these different compilers have a common internal structure. A per-language front end parses the source code in that language and produces an abstract syntax tree. If needed these are then

converted to the middle-end's input representation, called GENERIC form. The middle-end now gradually transforms the code towards its final form, by performing compiler optimizations and static code analysis techniques to the code. Finally, assembly language is produced using architecture-specific pattern matching.

The GCC is normally the compiler of choice for software developers to develop software which is meant to be capable of executing on a wide variety of hardware systems or operating systems. Unlike the system-specific compilers provided by hardware and operating system vendors, the code written for the GCC compiler is almost the same on every platform and only requires the platform-specific part of the code to be rewritten for each system.

GNU Compiler Collection has played an important role in the growth of free software both as a tool and as an example. These days for a processor architecture to become successful, the chip manufacturers consider it essential to have a port of GCC available for the architecture.

Microsoft Windows is a series of graphical user interface based software operating system developed by Microsoft. It is the most widely recognized operating system in the world and currently holds the majority share in the personal computer market. Currently Microsoft is making the Windows operating systems for mainly three platforms: the server (Windows Server 2008), the personal computer (Windows 7) and for the mobile phone platform (Windows Phone 7).

Microsoft originally started porting the Windows Operating System to 64-bit for the Intel's Itanium processor architecture and released IA64 versions of Windows 2000 Advance Server, Windows XP and Windows Server 2003. Later Microsoft shifted the 64-bit versions of its operating systems to x64 because the x64 architecture was backwards compatible with the x86 architecture and because the Itanium hardware was not easily available for development at that time. So far, Microsoft has released seventeen editions of its Windows Operating System for the x64 processor architecture.

Until the release of Windows XP in 2001 Microsoft used mainly two types of kernels, one for the windows 3.x and 9x family and the other for the Windows NT family, where the windows 3.x and 9.x were meant for the general user and the Windows NT was meant for the power users and for server use. Since the launch of Windows XP Microsoft has completely shifted all of its operating systems to the NT kernel.

Linux is a family of UNIX-like operating systems which use the Linux kernel. The name "Linux" comes from the Linux kernel, originally written in 1991 by *Linus Torvalds*. The Linux operating systems can be installed on a wide variety of hardware platforms ranging from supercomputers and mainframes to personal computers to videogame consoles to tablet computers and mobile phones.

The development of Linux is one of the most prominent examples of free and open source software collaboration. Generally, the entire source code can be redistributed both commercially and non-commercially by anyone under licenses such as the GNU General Public License after freely modifying it. A complete package of the Linux Operating system is known as a Linux distribution. The Linux distribution comprises the underlying Linux kernel with libraries and supporting utilities to fulfill the distribution's intended use.

The Linux system is a Unix-like modular operating system and derives its main design principles from the design principles established by the UNIX operating system. It uses a monolithic kernel called the Linux-kernel, which handles networking, control of processor and peripheral devices and file system access. The device drivers either are integrated directly into the kernel or are loaded as modules while the system is running.

There are multiple ways a user can operate a Linux-based system. Depending on the type of installation and the system it is installed on, the user can operate via a command line interface (CLI), a graphical user interface (GUI) or via the controllers attached to the hardware of the embedded system. For most desktop systems, the graphical user interface is the default mode even though the majority of low-level Linux components use the CLI exclusively. In the GUI based system, the CLI is provided through a terminal emulator window or by a separate virtual console. The CLI is typically implemented by a shell, which is

also the traditional way of interacting with a UNIX system. For tasks requiring automated repetitive or delayed work, CLI is better suited as it provides a simple inter-process communication. Sometimes the Linux distribution intended for server use only has the command line interface as its default.

The major difference between Linux and most of the other popular operating systems is that the Linux kernel and its components are free. Linux is not the only operating system whose kernel and components are free but it is the most widely used one. Many of these free and open source software licenses are based on the principle of copy-left which means any work derived from such a software must also be copy-left itself.

The Linux distributions are generally designed for use on desktop and server systems but they can be specialized for different purposes like computer architecture support, embedded systems, stability, security, localization to a specific region or language, targeting of specific user groups, support for real-time applications or commitment to a given desktop environment.

Linux is a widely ported operating system kernel. The Linux kernel runs on a highly diverse range of computer architectures, even on architectures that were only ever intended to use a manufacturer-created operating system.

Pthreads-w32 is a package, which on top of the existing Windows API provides a portable and open-source implementation of pthreads. The Win32 pthreads is normally implemented as a dynamic link library (DLL). Although this has some notable advantages from the Win32 point of view, it also closely models existing pthread libraries on UNIX, which are usually shared objects. It is also possible to build the library for static linking if necessary.

POSIX Threads commonly known as pthreads is a standard for threads which defines an application-programming interface (API) for creating and manipulating multithreaded applications. A number of modern day operating systems include support for this threading library.

This API defines a set of types, functions and constants that are implemented with the pthread.h header file and a thread library in the C programming language.

The Pthreads provides around 100 procedures all of which are prefixed with "pthread_". These different procedures can be categorized into the four groups: (i) Thread Management – The major concerns of this group of procedures are with the thread creation, joining, etc. (ii) Mutexes (iii) Condition variables (iv) Synchronization - This group of procedures are concerned with the synchronization between threads using read/write locks and barriers.

Testing Routine

The accuracy and reliability of the results of simulations performed by the newly created *PopSim* program and the speed of execution of the simulations were tested using one artificial example simulation and eight marker-assisted backcrossing examples which were originally used in creation of Table 3 of '*Selection strategies for marker-assisted backcrossing with high-throughput marker system*' (*Herzog and Frisch 2011*). Two QTL mapping examples, which were used in Figure 2 of '*A comparison of tests for QTL mapping with introgression libraries containing overlapping and non-overlapping donor segments.' (Mahone et al. 2012)* were also used for testing of the software. The artificial example simulated the offspring in Mendel's experiment where several differentiating characters are associated by taking three characters i.e. form of seed, color of albumen and color of seed coat. The code of this script can be found in *Appendix II*. The marker-assisted backcrossing examples used for testing have been given below.

1. 2cM Equally: It was used in Table 3 of '*Selection strategies for marker-assisted backcrossing with high-throughput marker system*' (*Herzog and Frisch 2011*). This simulation gave Q10 (10% quantile) values recovered in the 3rd generation of back crossing (BC3) for constant population sizes in 1st generation backcrossing (BC1), 2nd generations backcrossing (BC2) and BC3 for equally spaced marker 2cM by applying two-stage selection with High-throughput (HT) assays. The code of this script has been provided in *Appendix III*.
2. 2cM Randomly: It was used by *Herzog and Frisch (2011)* in Table 3 of '*Selection strategies for marker-assisted backcrossing with high-throughput marker system*'. This simulation gave the Q10 values recovered in BC3 for constant population sizes,

from BC1 to BC3 for randomly distributed marker 2cM by applying two-stage selection with HT assays.

3. 5cM Equally: It was used by *Herzog and Frisch (2011)* in Table 3 of '*Selection strategies for marker-assisted backcrossing with high-throughput marker system*'. This simulation gave the Q10 values recovered in BC3 for constant population sizes, from BC1 to BC3 for equally spaced marker 5cM by applying two-stage selection with HT assays.
4. 5cM Randomly: It was used in Table 3 of '*Selection strategies for marker-assisted backcrossing with high-throughput marker system*' (*Herzog and Frisch 2011*). This simulation gave the Q10 values recovered in BC3 for constant population sizes, from BC1 to BC3 for randomly distributed marker 5cM by applying two-stage selection with HT assays.
5. 10cM Equally: It was used in Table 3 of '*Selection strategies for marker-assisted backcrossing with high-throughput marker system*' (*Herzog and Frisch 2011*). This simulation gave the Q10 values recovered in BC3 for constant population sizes, from BC1 to BC3 for equally spaced marker 10cM by applying two-stage selection with HT assays.
6. 10cM Randomly: It was used by *Herzog and Frisch (2011)* in Table 3 of '*Selection strategies for marker-assisted backcrossing with high-throughput marker system*'. This simulation gave the Q10 values recovered in BC3 for constant population sizes, from BC1 to BC3 for randomly distributed marker 10cM by applying two-stage selection with HT assays.
7. 20cM Equally: It was used in Table 3 of '*Selection strategies for marker-assisted backcrossing with high-throughput marker system*' (*Herzog and Frisch 2011*). This simulation gave the Q10 values recovered in BC3 for constant population sizes, from BC1 to BC3 for equally spaced marker 20cM by applying two-stage selection with HT assays.
8. 20cM Randomly: It was used by *Herzog and Frisch (2011)* in Table 3 of 'Selection strategies for marker-assisted backcrossing with high-throughput marker system'. This simulation gave the Q10 values recovered in BC3 for constant population sizes, from BC1 to BC3 for randomly distributed marker 20cM by applying two-stage selection with HT assays.

The QTL mapping examples used for testing have been given below.

1. IL-Tests-p10: It was used in Figure 2 of '*A comparison of tests for QTL mapping with introgression libraries containing overlapping and non-overlapping donor segments.' (Mahone et al. 2012)* simulated quantitative trait loci (QTL) in NIL lines (Interogression libraries) and attempts to detect them using a linear model.
2. Overlap-DTrun-MSeg-np: It was used in Figure 2 of '*A comparison of tests for QTL mapping with introgression libraries containing overlapping and non-overlapping donor segments.' (Mahone et al. 2012)* simulated QTL in NIL lines along with an attempt to detect them using the "Dunnett test".

The recording of the time taken by the simulations under both *PopSim* and *Plabsoft* a small piece of code was added to each script. This code saved the system time in a variable, after loading of the respective packages was completed, then after finishing of the simulation this code would subtract the system time saved at the beginning of the simulation from the current system time. The reason why the timer was started after loading the packages was to make sure that the loading time of each package had no effect on the time taken by the simulation.

Since the load-time of the packages was not measured during the comparison tests. Another set of tests were performed where each package was loaded three times and the time taken by the package to load was measured using the same method used to record the simulation time.

The time measurements were only carried out on the Linux System. The specifications of the Linux System that was used for the development and testing of *PopSim* were *Intel Core 2 Quad 2.67 GHz* Processor with 4 GB of RAM.

3. Results

One of the major requirements for *PopSim* was that its results should be comparable to the results of *Plabsoft* and *PopSim* should be capable of running scripts written for *Plabsoft* without any changes to the script. Therefore, after the coding of *PopSim* was completed, it was tested against *Plabsoft* using one artificial example script. The two software programs were also compared using eight marker-assisted backcrossing examples and two QTL mapping examples. The testing was done primarily to confirm that the results of *PopSim* were on par with the results produced by *Plabsoft* and to verify that *PopSim* is truly faster than *Plabsoft* for the scripts that use the multithreaded part of the *PopSim*. Another reason was to confirm that *PopSim* is not slower than *Plabsoft* for scripts that do not use the multithreaded part of *PopSim*.

In addition to these tests another set of tests were performed to compare the results of *PopSim* on Windows machine to the results of *PopSim* on Linux machine to make sure that the output of *PopSim* on both the systems is identical.

The artificial example used for testing simulated the Mendel's experiment for three characters. All of the marker-assisted backcrossing and QTL mapping examples were tested at the actual repetitions used in the original simulation on both *PopSim* and *Plabsoft* on the Linux machine. Along with running at the original repetitions, both of the QTL mapping examples and two of the marker-assisted backcrossing examples were run at reduced repetitions as well. The following marker-assisted backcrossing examples were tested both at reduced and at original repetitions used in the original experiment:

1. 2cM Equally
2. 2cM Randomly

Below are the Marker-assisted backcrossing examples, which were simulated only once, on the Linux machine at original repetitions by both *PopSim* and *Plabsoft*:

1. 5cM Equally
2. 5cM Randomly
3. 10cM Equally
4. 10cM Randomly
5. 20cM Equally
6. 20cM Randomly

3.1 Load Time Comparison of PopSim and Plabsoft

The documenting of the time engaged by *Plabsoft* and *PopSim* to load was done by storing the system time in a variable just before loading the package. This time stored in variable was subtracted from the system time after the loading of the package was completed. The difference between the two times was the actual time taken by the package to load. This entire process of measuring the loading time was repeated three times for both *PopSim* and *Plabsoft*. On matching up the time taken by each program it was discovered that *PopSim* was taking only 1/3rd of the time that *Plabsoft* was taking on average, which means that typically *PopSim* took 0.082289366 seconds less than *Plabsoft*. The recordings from all the three runs performed to measure the load times are illustrated in Table 1.

3.2 Comparison of Results from Artificial Example

The artificial example was run under different scenarios (using different combinations of population sizes and repetitions) and the results from these simulations from both *PopSim* and *Plabsoft* are compared.

3.2.1 Artificial Example Scenario 1

In this scenario, the Artificial Example was run with a Population size of 10 and the repetitions were set to 100. The results from both *PopSim* and *Plabsoft* are similar in this scenario. The detailed comparison can be seen in Table 2.

3.2.2 Artificial Example Scenario 2

In this case, the Artificial Example was executed with the Population size set to 100 and the repetitions were set to 100. On comparing the results from both *PopSim* and *Plabsoft*, it was noted that both software gave similar results. The detailed comparison of these results is presented in Table 3.

Table 1: Comparison of *Plabsoft* & *PopSim* Load Time in Seconds

Measurement	*Plabsoft*	*PopSim*
1	0.128587	0.04275703
2	0.1253252	0.04146004
3	0.120064	0.04289103
Average	0.124658733	0.042369367

Table 2: Comparison of Results of Artificial Example with 100 Repetitions and Population Size of10

Plabsoft		*PopSim* on Linux		*PopSim* on Windows	
	Results		Results		Results
fo - 1	8.000	fo - 1	8.000	fo - 1	8.000
fo - 2	8.000	fo - 2	8.000	fo - 2	8.000
al - 1	8.000	al - 1	8.000	al - 1	8.000
al - 2	8.000	al - 2	8.000	al - 2	8.000
sc - 1	8.000	sc - 1	8.000	sc - 1	8.000
sc - 2	8.000	sc - 2	8.000	sc - 2	8.000
Count	10.00	Count	10.00	Count	11.000
Frequency	0.01	Frequency	0.01	Frequency	0.011

(fo = Form of seed, al = Colour of albumen and sc = Colour of seed coats.)

3.2.3 Artificial Example Scenario 3

For this scenario, the population size was changed to 1000 and the repetitions to 100, while simulating the Artificial Example. The results produced by both the *PopSim* and *Plabsoft* were comparable. Table 4 shows the detailed matchup of the results of this simulation.

3.2.4 Artificial Example Scenario 4

In case of this scenario, the Artificial Example was simulated by both *Plabsoft* and *PopSim* with the population size of 10000 and 100 repetitions. It was found that both *PopSim* and *Plabsoft* produced similar results, which can be studied in the Table 5.

3.2.5 Artificial Example Scenario 5

For the fifth scenario, the Artificial Example was simulated with the Population size set to 10 and repetitions set to 1000 on both *PopSim* and *Plabsoft*. By comparing, the results shown in Table 6 it was noted that both of the software produced similar results.

3.2.6 Artificial Example Scenario 6

A population size of 100 with 1000 repetitions was used in this scenario for simulating the Artificial Example. For this scenario, both the *PopSim* and *Plabsoft* produced comparable results, which are displayed in detail in the Table 7.

Table 3: Comparison of Results of Artificial Example with 100 Repetitions and Population Size of 100

Plabsoft		*PopSim* on Linux		*PopSim* on Windows	
	Results		Results		Results
fo - 1	8.0000	fo - 1	8.0000	fo - 1	8.0000
fo - 2	8.0000	fo - 2	8.0000	fo - 2	8.0000
al - 1	8.0000	al - 1	8.0000	al - 1	8.0000
al - 2	8.0000	al - 2	8.0000	al - 2	8.0000
sc - 1	8.0000	sc - 1	8.0000	sc - 1	8.0000
sc - 2	8.0000	sc - 2	8.0000	sc - 2	8.0000
Count	175.0000	Count	158.0000	Count	154.0000
Frequency	0.0175	Frequency	0.0158	Frequency	0.0154

(fo = Form of seed, al = Colour of albumen and sc = Colour of seed coats.)

Table 4: Comparison of Results of Artificial Example with 100 Repetitions and Population Size of 1000

Plabsoft		*PopSim* on Linux		*PopSim* on Windows	
	Results		Results		Results
fo - 1	8.000e+00	fo - 1	8.000e+00	fo - 1	8.000e+00
fo - 2	8.000e+00	fo - 2	8.000e+00	fo - 2	8.000e+00
al - 1	8.000e+00	al - 1	8.000e+00	al - 1	8.000e+00
al - 2	8.000e+00	al - 2	8.000e+00	al - 2	8.000e+00
sc - 1	8.000e+00	sc - 1	8.000e+00	sc - 1	8.000e+00
sc - 2	8.000e+00	sc - 2	8.000e+00	sc - 2	8.000e+00
Count	1.579e+03	Count	1.573e+03	Count	1.504e+03
Frequency	1.579e-02	Frequency	1.573e-02	Frequency	1.504e-02

(fo = Form of seed, al = Colour of albumen and sc = Colour of seed coats.)

Table 5: Comparison of Results of Artificial Example with 100 Repetitions and Population Size of 10000

Plabsoft		*PopSim* on Linux		*PopSim* on Windows	
	Results		Results		Results
fo - 1	8.0000e+00	fo - 1	8.0000e+00	fo – 1	8.0000e+00
fo - 2	8.0000e+00	fo - 2	8.0000e+00	fo – 2	8.0000e+00
al - 1	8.0000e+00	al - 1	8.0000e+00	al – 1	8.0000e+00
al - 2	8.0000e+00	al - 2	8.0000e+00	al – 2	8.0000e+00
sc - 1	8.0000e+00	sc - 1	8.0000e+00	sc – 1	8.0000e+00
sc - 2	8.0000e+00	sc - 2	8.0000e+00	sc – 2	8.0000e+00
Count	1.5614e+04	Count	1.5809e+04	Count	1.5461e+04
Frequency	1.5614e-02	Frequency	1.5809e-02	Frequency	1.5461e-02

(fo = Form of seed, al = Colour of albumen and sc = Colour of seed coats.)

Table 6: Comparison of Results of Artificial Example with 1000 Repetitions and Population Size of 10

Plabsoft		*PopSim* (on Linux)		*PopSim* (on Windows)	
	Results		Results		Results
fo - 1	8.0000	fo - 1	8.0000	fo - 1	8.0000
fo - 2	8.0000	fo - 2	8.0000	fo - 2	8.0000
al - 1	8.0000	al - 1	8.0000	al - 1	8.0000
al - 2	8.0000	al - 2	8.0000	al - 2	8.0000
sc - 1	8.0000	sc - 1	8.0000	sc - 1	8.0000
sc - 2	8.0000	sc - 2	8.0000	sc - 2	8.0000
Count	169.0000	Count	160.0000	Count	161.0000
Frequency	0.0169	Frequency	0.016	Frequency	0.0161

(fo = Form of seed, al = Colour of albumen and sc = Colour of seed coats.)

3.2.7 Artificial Example Scenario 7

The Artificial Example was simulated in this scenario with a population size of 1000 and 1000 repetitions. After studying the results revealed in Table 8 from both the *Plabsoft* and *PopSim*, it was found that both of these software produced similar output.

Finally on examining Tables 2-8 it was found that as far as the Artificial Example is concerned the output from both the *PopSim* and *Plabsoft* was on similar grounds irrespective of whether *PopSim* was run on the Windows or Linux machine.

3.3 Simulation Time Comparison of Artificial Example

To check the speed difference between *PopSim* and *Plabsoft* in the execution of the Artificial Example, the Artificial Example was simulated three times for each scenario on both *PopSim* and *Plabsoft* alternately on the Linux Machine. The time taken by both of these software to simulate the Artificial Example under different scenarios was compared.

3.3.1 Artificial Example Scenario 1

In this scenario, the Artificial Example was simulated with a Population size of 10 and the repetitions were set to 100. It was noted that on average *PopSim* took 22% less time (0.0146977 Seconds) than *Plabsoft* for this scenario.

The Table 9 shows the detailed comparison of the time taken by *PopSim* and *Plabsoft* for the three alternative runs of the simulation.

3.3.2 Artificial Example Scenario 2

In case of this scenario, the Population size was changed to 100 and the repetitions to 100 while simulating the Artificial Example. After recording the execution times of both *PopSim* and *Plabsoft* for each of the three runs, it was observed that *PopSim* only took approximately 2/3rd the time of *Plabsoft* on average or 0.03637303 seconds (33%) less than *Plabsoft*.

The details of time taken by each run of the simulation on both *PopSim* and *Plabsoft* for this scenario are displayed in the Table 10.

Table 7: Comparison of Results of Artificial Example with 1000 Repetitions and Population Size of 100

Plabsoft		*PopSim* on Linux		*PopSim* on Windows	
	Results		Results		Results
fo - 1	8.000e+00	fo - 1	8.000e+00	fo - 1	8.000e+00
fo - 2	8.000e+00	fo - 2	8.000e+00	fo - 2	8.000e+00
al - 1	8.000e+00	al - 1	8.000e+00	al - 1	8.000e+00
al - 2	8.000e+00	al - 2	8.000e+00	al - 2	8.000e+00
sc - 1	8.000e+00	sc - 1	8.000e+00	sc - 1	8.000e+00
sc - 2	8.000e+00	sc - 2	8.000e+00	sc - 2	8.000e+00
Count	1.525e+03	Count	1.518e+03	Count	1.508e+03
Frequency	1.525e-02	Frequency	1.518e-02	Frequency	1.508e-02

(fo = Form of seed, al = Colour of albumen and sc = Colour of seed coats.)

Table 8: Comparison of Results of Artificial Example with 1000 Repetitions and Population Size of 1000

Plabsoft		*PopSim* on Linux		*PopSim* on Windows	
	Results		Results		Results
fo - 1	8.0000e+00	fo - 1	8.0000e+00	fo - 1	8.0000e+00
fo - 2	8.0000e+00	fo - 2	8.0000e+00	fo - 2	8.0000e+00
al - 1	8.0000e+00	al - 1	8.0000e+00	al - 1	8.0000e+00
al - 2	8.0000e+00	al - 2	8.0000e+00	al - 2	8.0000e+00
sc - 1	8.0000e+00	sc - 1	8.0000e+00	sc - 1	8.0000e+00
sc - 2	8.0000e+00	sc - 2	8.0000e+00	sc - 2	8.0000e+00
Count	1.5554e+04	Count	1.5573e+04	Count	1. 5446e+04
Frequency	1.5554e-02	Frequency	1.5573e-02	Frequency	1. 5446e-02

(fo = Form of seed, al = Colour of albumen and sc = Colour of seed coats.)

Table 9: Comparison of Time of Artificial Example with 100 Repetitions and Population Size of 10 in Seconds

Run	*Plabsoft*	*PopSim*
1	0.072973	0.050936
2	0.056967	0.050936
3	0.066968	0.050943
Average	0.065636	0.0509383

3.3.3 Artificial Example Scenario 3

For the third scenario, the Artificial Example was simulated on both the *PopSim* and *Plabsoft* with a population size of 1000 and repetitions set to 100. On average *PopSim* took 0.21578 Seconds (40%) less than *Plabsoft* for carrying out of this simulation.

The complete comparisons of the time employed by both the *PopSim* and *Plabsoft* for the three runs of this simulation are presented in the Table 11.

Table 10: Comparison of Time of Artificial Example
with 100 Repetitions and Population Size of 100 in Seconds

Run	*Plabsoft*	*PopSim*
1	0.10295	0.073923
2	0.10397	0.072902
3	0.12097	0.071946
Average	0.1092967	0.07292367

Table 11: Comparison of Time of Artificial Example
with 100 Repetitions and Population Size of 1000 in Seconds

Run	*Plabsoft*	*PopSim*
1	0.55295	0.34095
2	0.54219	0.33495
3	0.55318	0.32508
Average	0.54944	0.33366

3.3.4 Artificial Example Scenario 4

In this case, the Artificial Example was simulated at 100 repetitions with a Population size of 10000 on both of the software. In the execution of this scenario, *PopSim* on average used 2.198867 Seconds (42%) less computer time than *Plabsoft*.

The time engaged by both the *PopSim* and *Plabsoft* for the three simulation runs of this scenario are exhibited in the Table 12.

3.3.5 Artificial Example Scenario 5 (Repetitions 1000 with Population Size 10)

The Artificial Example was simulated with a population size of 10 and 1000 repetitions for this scenario. In case of this scenario the use of *PopSim* provided time savings of 0.0389633 seconds (16%) over the use of *Plabsoft*.

The Table 13 shows the detailed evaluation of the computer time consumed by each of the three execution runs of the simulation on both the *PopSim* and *Plabsoft*.

Table 12: Comparison of Time of Artificial Example
with 100 Repetitions and Population Size of 10000 in Seconds

Run	*Plabsoft*	*PopSim*
1	5.2224	3.0074
2	5.2161	3.0363
3	5.2481	3.0463
Average	5.228867	3.03

Table 13: Comparison of Time of Artificial Example
with 1000 Repetitions and Population Size of 10 in Seconds

Run	*Plabsoft*	*PopSim*
1	0.26495	0.20299
2	0.23101	0.20398
3	0.23035	0.20245
Average	0.2421033	0.20314

3.3.6 Artificial Example Scenario 6

For the sixth scenario, the Artificial Example was executed at 1000 repetitions with a Population size of 100 on both *PopSim* and *Plabsoft*. *PopSim* on average utilized 0.1456733 Seconds (18%) less time than *Plabsoft* to finish this simulation.

The complete comparison between the time taken by each of the three runs of the simulation on *PopSim* and *Plabsoft* is displayed in the Table 14.

3.3.7 Artificial Example Scenario 7 (Repetitions 1000 with Population Size 1000)

A population size of 1000 was used for simulating of the Artificial Example in scenario 7 with the repetitions set to 1000. For the execution of this simulation, *PopSim* required 2.135233 Seconds (22%) less than *Plabsoft* in this scenario.

The Table 15 presents the details of the time taken by each of the three simulation runs on both *PopSim* and *Plabsoft*.

Table 14: Comparison of Time of Artificial Example
with 1000 Repetitions and Population Size of 100 in Seconds

Run	*Plabsoft*	*PopSim*
1	0.80199	0.66297
2	0.81789	0.68721
3	0.80697	0.63965
Average	0.80895	0.6632767

Table 15: Comparison of Time of Artificial Example
with 1000 Repetitions and Population Size of 1000 in Seconds

Run	*Plabsoft*	*PopSim*
1	9.542	7.3651
2	9.5241	7.3472
3	9.407	7.3551
Average	9.491033	7.3558

On examining the tables Table 9 to Table 15, it was observed that two patterns were emerging. Firstly it was noted that on increasing the size of the population the difference between the time taken by *PopSim* & *Plabsoft* to perform the simulation amplified, which makes sense because as the size of the population increases so does the portion of the simulation which can be parallelized. Additionally, the percentage of time wasted on creation, switching and controlling the threads decreases as the population grows. The second pattern that was observed was that on increasing the number of repetitions the difference between the time used by *PopSim* and *Plabsoft* to carry out the simulation diminished, the reason for this is that the repetition loop was not controlled by the *PopSim* but by the example script. Therefore, even though the code executed by *PopSim* was parallelized, the code of the repetition loop was not parallelized because the repetition loop was a sequential code outside of *PopSim*.

3.4 Comparison of Results from Marker-Assisted Backcrossing Examples

3.4.1 Results of 2cM Equally Simulation

3.4.1.1 Repetitions reduced to 100

The output of *PopSim* simulations from both the Linux System and the Windows System were on the same grounds as that of the *Plabsoft* for this scenario where the '2cM equally' simulation was executed with repetitions reduced to 100 instead of the original 10000.

The output from the *Plabsoft* simulation is shown in the Table 16 while the Table 17 and Table 18 display the results from the *PopSim* Simulation on the Linux and Windows machine respectively.

Table 16: Results of *Plabsoft* Simulation of '*2cM Equally*' with reduced repetitions of 100

Generation	Equally spaced markers, n_t								
	40	60	80	100	120	140	160	180	200
BC1	79.7	80.7	82.1	82.3	82.8	83.4	83.3	83.7	83.7
BC2	92.7	94.1	94.3	95.1	95.3	95.5	95.9	96.1	95.9
BC3	97.7	98.3	98.5	98.8	98.7	99.0	99.1	99.2	99.1

(Q10 values recovered in generation BC3 for Constant population size n_t in generations BC1 to BC3 and equally spaced marker 2cM applying two-stage selection with HT assays)

Table 17: Results of '*2cM Equally*' Simulation by *PopSim* on Linux with reduced repetitions of 100

Generation	Equally spaced markers, n_t								
	40	60	80	100	120	140	160	180	200
BC1	80.2	81.6	81.9	82.3	83.2	82.8	83.3	84.2	84.0
BC2	93.4	94.2	94.5	95.0	95.4	95.7	95.4	96.0	96.0
BC3	97.8	98.2	98.5	98.6	98.9	99.0	99.0	99.1	99.1

(Q10 values recovered in generation BC3 for Constant population size n_t in generations BC1 to BC3 and equally spaced marker 2cM applying two-stage selection with HT assays)

Table 18: Results of '*2cM Equally*' Simulation by *PopSim* on Windows with reduced repetitions of 100

Generation	Equally spaced markers, n_t								
	40	60	80	100	120	140	160	180	200
BC1	79.8	80.8	82.1	82.2	83.2	83.3	83.3	83.8	84.2
BC2	93.1	93.9	94.6	95.0	95.5	95.5	95.7	96.1	95.9
BC3	97.8	98.0	98.5	98.6	98.8	98.8	98.9	99.1	99.0

(Q10 values recovered in generation BC3 for Constant population size n_t in generations BC1 to BC3 and equally spaced marker 2cM applying two-stage selection with HT assays)

3.4.1.2 Repetitions set to original 10000:

The repetitions of the '2cM Equally' simulation were set to the original repetitions of 10000 which were used in the original experiment and the simulation was run on both the software. After examining the results of the simulation run by the two software it was observed that the outputs of the both the *PopSim* and *Plabsoft* were identical.

The Table 19 displays the output of the '2cM Equally' simulation.

On examining the tables 'Table 16' to 'Table 19' it was observed that both the *PopSim* and *Plabsoft* present similar outcome for the '2cM Equally' simulation irrespective of whether the simulation was run at the original repetitions or at reduced repetitions.

Table 19: Results of '*2cM Equally*' Simulation with 10000 Repetitions

Generation	Equally spaced markers, n_t								
	40	60	80	100	120	140	160	180	200
BC1	80.0	81.1	81.8	82.3	82.8	83.1	83.4	83.6	83.8
BC2	93.2	94.0	94.6	95.0	95.3	95.5	95.7	95.8	96.0
BC3	97.8	98.2	98.5	98.7	98.8	98.9	99.0	99.0	99.1

(Q10 values recovered in generation BC3 for Constant population size n_t in generations BC1 to BC3 and equally spaced marker 2cM applying two-stage selection with HT assays)

3.4.2 Results of 2cM Randomly Simulation

3.4.2.1 Repetitions reduced to 100:

The '2cM Randomly' simulation was executed alternatively on both the *PopSim* and *Plabsoft* with repetitions reduced to 100 instead of the original 10000. It was observed on investigating the results from the simulations that both the software produced similar outcome irrespective of whether the *PopSim* was run on the Linux or the Windows Machine.

The results of the *Plabsoft* simulation are shown in the Table 20 and that of the *PopSim* simulations are shown in Table 21 for the Linux System and in the Table 22 for the Windows System.

Table 20: Results of *Plabsoft* Simulation of '*2cM Randomly*' with reduced repetitions of 100

Generation	Equally spaced markers, n_t								
	40	60	80	100	120	140	160	180	200
BC1	79.8	80.9	82.2	82.1	82.9	82.6	83.0	83.5	83.9
BC2	93.2	93.8	94.7	95.2	95.4	95.3	95.3	95.8	95.7
BC3	97.5	98.2	98.5	98.8	98.9	98.7	98.9	99.0	99.0

(Q10 values recovered in generation BC3 for Constant population size n_t in generations BC1 to BC3 and randomly distributed marker 2cM applying two-stage selection with HT assays)

Table 21: Results of Simulation of '*2cM Randomly*' with *PopSim* on Linux using reduced repetitions of 100

Generation	Equally spaced markers, n_t								
	40	60	80	100	120	140	160	180	200
BC1	80.9	80.8	81.4	82.4	83.0	83.0	83.2	83.5	83.8
BC2	92.8	93.6	94.8	95.0	95.1	95.2	95.8	95.7	95.8
BC3	97.9	98.1	98.4	98.6	98.8	99.1	99.1	99.1	99.2

(Q10 values recovered in generation BC3 for Constant population size n_t in generations BC1 to BC3 and randomly distributed marker 2cM applying two-stage selection with HT assays)

Table 22: Results of Simulation of '*2cM Randomly*' with *PopSim* on Windows using reduced repetitions of 100

Generation	Equally spaced markers, n_t								
	40	60	80	100	120	140	160	180	200
BC1	79.9	81.1	81.3	82.4	82.6	82.9	83.1	83.2	83.0
BC2	92.7	93.9	94.3	94.7	95.0	95.3	95.5	95.7	95.5
BC3	97.7	98.2	98.4	98.7	98.7	98.8	98.9	99.0	99.0

(Q10 values recovered in generation BC3 for Constant population size n_t in generations BC1 to BC3 and randomly distributed marker 2cM applying two-stage selection with HT assays)

3.4.2.2 Repetitions set to original 10000:

For this scenario, the repetitions for the '2cM Randomly' simulation were set to the original repetitions of 10000 which were used in the original experiment and the script was run on both the *Plabsoft* and *PopSim* alternatively. For this setting both the *PopSim* and *Plabsoft* gave matching results.

The outcome of the simulation is revealed in the Table 23.

By studying the tables Table 20 to Table 23 it was noticed that both the *PopSim* and *Plabsoft* were producing similar output for the '2cM Randomly' simulation, irrespective of whether the simulation was executed at its original repetitions or at reduced repetitions.

Table 23: Results of '*2cM Randomly*' Simulation with 10000 Repetitions

Generation	Equally spaced markers, n_t								
	40	60	80	100	120	140	160	180	200
BC1	79.8	80.9	81.5	82.1	82.5	82.8	83.1	83.4	83.7
BC2	93.0	93.9	94.4	94.8	95.1	95.3	95.5	95.7	95.9
BC3	97.7	98.2	98.5	98.6	98.7	98.9	98.9	99.0	99.1

(Q10 values recovered in generation BC3 for Constant population size n_t in generations BC1 to BC3 and randomly distributed marker 2cM applying two-stage selection with HT assays)

3.4.3 Results of 5cM Equally Simulation

After executing the '5cM Equally' simulation on both the *PopSim* and *Plabsoft* at 10000 repetitions, the same number of repetitions used in the original experiment. It was found out that the outcome of the ´simulation performed by *PopSim* was exactly the same as the output of the *Plabsoft* simulation.

The Table 24 illustrates the results of the '5cM Equally' simulation carried out on both the *Plabsoft* and *PopSim*.

3.4.5 Results of 5cM Randomly Simulation

In this case, the '5cM Randomly' simulation was performed by both *PopSim* and *Plabsoft* at the original repetitions of 10000 employed by the original experiment. Here the output of the simulation executed by *PopSim* was identical to the results of the simulation run on the *Plabsoft*.

The output of the '5cM Randomly' simulation done by both the *Plabsoft* and *PopSim* is shown in the Table 25.

Table 24: Results of '*5cM Equally*' Simulation with 10000 Repetitions

Generation	Equally spaced markers, n_t								
	40	60	80	100	120	140	160	180	200
BC1	80.0	81.1	81.7	82.3	82.7	83.0	83.4	83.6	83.9
BC2	93.1	94.0	94.5	94.9	95.3	95.4	95.7	95.8	96.0
BC3	97.8	98.2	98.5	98.6	98.8	98.9	99.0	99.0	99.1

(Q10 values recovered in generation BC3 for Constant population size n_t in generations BC1 to BC3 and equally spaced marker 5cM applying two-stage selection with HT assays)

Table 25: Results of '*5cM Randomly*' Simulation with 10000 Repetitions

Generation	Equally spaced markers, n_t								
	40	60	80	100	120	140	160	180	200
BC1	79.3	80.4	81.0	81.5	81.9	82.3	82.5	82.8	83.0
BC2	92.4	93.3	93.8	94.2	94.4	94.6	94.8	95.0	95.1
BC3	97.1	97.6	97.9	98.1	98.3	98.4	98.4	98.5	98.6

(Q10 values recovered in generation BC3 for Constant population size n_t in generations BC1 to BC3 and randomly distributed marker 5cM applying two-stage selection with HT assays)

3.4.6 Results of 10cM Equally Simulation

The simulation '10cM Equally' was carried out on both *PopSim* and *Plabsoft* at 10000 repetitions the same number of repetitions that were used in the original experiment. On comparing the outputs of the two software, it was noted that the outcomes of the simulation run from both the *PopSim* and *Plabsoft* were on par with each other.

The Table 26 displays the end result of the '10cM Equally' simulation run on *Plabsoft* as well as the *PopSim*.

3.4.7 Results of 10cM Randomly Simulation:

The number repetitions was set 10000 the same number employed by the original experiment for the '10cM Randomly' simulation. This simulation was performed on both *Plabsoft* and *PopSim* alternatively. On execution of the simulation, it was found out that the results of both the *PopSim* and *Plabsoft* were identical to one another.

The results from the simulation of '10cM Randomly' are illustrated in the Table 27.

3.4.8 Results of 20cM Equally Simulation

The '20cM Equally' simulation was executed alternatively on both the *Plabsoft* and *PopSim* with the repetitions set to 10000 (The number of repetitions used in the original simulation). On comparing the output from both of the simulation runs, it was observed that both *PopSim* and *Plabsoft* were producing results identical to each other.

The outcome of the '20cM Equally' simulation done by both the *Plabsoft* and *PopSim* is shown in the Table 28.

Table 26: Results of '*10cM Equally*' Simulation with 10000 Repetitions

Generation	Equally spaced markers, n_t								
	40	60	80	100	120	140	160	180	200
BC1	79.9	81.0	81.7	82.3	82.7	83.0	83.3	83.5	83.8
BC2	93.0	93.9	94.5	94.9	95.2	95.4	95.6	95.7	95.9
BC3	97.6	98.1	98.4	98.6	98.7	98.8	98.9	98.9	99.0

(Q10 values recovered in generation BC3 for Constant population size n_t in generations BC1 to BC3 and equally spaced marker 10cM applying two-stage selection with HT assays)

Table 27: Results of '*10cM Randomly*' Simulation with 10000 Repetitions

Generation	Equally spaced markers, n_t								
	40	60	80	100	120	140	160	180	200
BC1	78.8	79.8	80.5	80.9	81.3	81.7	81.9	82.1	82.3
BC2	91.9	92.8	93.4	93.8	94.1	94.3	94.4	94.7	94.8
BC3	97.0	97.5	97.8	98.0	98.1	98.2	98.3	98.3	98.4

(Q10 values recovered in generation BC3 for Constant population size n_t in generations BC1 to BC3 and randomly distributed marker 10cM applying two-stage selection with HT assays)

Table 28: Results of '*20cM Equally*' Simulation with 10000 Repetitions

Generation	Equally spaced markers, n_t								
	40	60	80	100	120	140	160	180	200
BC1	79.7	80.8	81.4	82.0	82.4	82.7	83.0	83.2	83.4
BC2	92.8	93.7	94.2	94.6	94.9	95.1	95.3	95.4	95.6
BC3	97.4	97.9	98.1	98.3	98.4	98.5	98.6	98.6	98.7

(Q10 values recovered in generation BC3 for Constant population size n_t in generations BC1 to BC3 and equally spaced marker 20cM applying two-stage selection with HT assays)

Table 29: Results of *Plabsoft* Simulation of '*20cM Randomly*' with 10000 Repetitions

Generation	Equally spaced markers, n_t								
	40	60	80	100	120	140	160	180	200
BC1	78.0	78.9	79.6	80.0	80.5	80.7	80.9	81.2	81.4
BC2	91.3	92.1	92.6	92.9	93.2	93.5	93.6	93.8	94.0
BC3	96.4	96.9	97.0	97.2	97.3	97.4	97.4	97.5	97.5

(Q10 values recovered in generation BC3 for Constant population size n_t in generations BC1 to BC3 and randomly distributed marker 20cM applying two-stage selection with HT assays)

3.4.9 Results of 20cM Randomly Simulation

The simulation of the '20cM Randomly' was carried out with repetitions set to 10000 (The number of repetitions employed in the original experiment) on both the *Plabsoft* and *PopSim* alternatively. It was noticed on evaluating the outcomes from the two software that both the *PopSim* and *Plabsoft* were producing indistinguishable results.

The Table 29 is showing the results given by both the *Plabsoft* and *PopSim* after executing the '20cM Randomly'.

3.5 Comparison of Results from QTL Mapping Examples

3.5.1 Results of IL-Tests-p10 Simulation

3.5.1.1 Repetitions reduced to 50

For this situation, the 'IL-Tests-p10' simulation was performed alternatively on both *PopSim* and *Plabsoft* with the repetitions reduced to 50 from the original 5000. The *PopSim* simulations were done on both the Linux and the Windows machines. On completion of the simulations, the outputs from both the *Plabsoft* and the two *PopSim* runs were compared. It was observed that the results of all three simulations were alike.

The output of the 'IL-Tests-p10' simulation done on *Plabsoft* is shown in the Table 30 while the Table 31 and Table 32 displays the results of the two simulations performed by the *PopSim* on Linux and Windows system respectively.

3.5.1.2 Repetitions set to original 5000

In case of this set-up, the repetitions of the 'IL-Tests-p10' simulation were set to their original value of 5000. This simulation was executed on both the *PopSim* and *Plabsoft* alternatively and the outputs from both the simulations were compared. It was observed that both of the softwares were producing analogous results.

The Table 33 shows the *Plabsoft*'s outcome of the 'IL-Tests-p10' simulation while on the other hand Table 34 displays the *PopSim* results of the same simulation.

3.5.2 Results of Overlap-DTrun-MSeg-np Simulation

3.5.2.1 Repetitions reduced to 50:

The repetitions for the 'Overlap-DTrun-MSeg-np' simulation were reduced to 50 from the original 5000 for this setting. The simulation was the performed by both *Plabsoft* and *PopSim* on the Linux machine while it was only simulated by *PopSim* on the Windows machine. After the completion of all the three runs of the simulation, it was observed that the results produced by all the three executions of the simulations were similar to each other.

The outcome of the 'Overlap-DTrun-MSeg-np' simulation performed by the *Plabsoft* is displayed in the Table 35 and that of the *PopSim* are shown in the Table 36 and Table 37 for Linux and Windows runs respectively.

3.5.2.2 Repetitions set to original 5000

The 'Overlap-DTrun-MSeg-np' simulation was run at its original repetitions of 5000 on both the software on the Linux System. After studying the results of the simulation run on *Plabsoft* and *PopSim*, it was noted that the output given by *PopSim* was on par to that of *Plabsoft*.

Table 30: Results of *Plabsoft* Simulation of '*IL-Tests-p10*' with Repetitions Reduced to 50

S003-007	MA 3	MI 0	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	47.1	46.8	0	0.28	46.5
0.6	0.6	79.2	78.6	0	0.64	64.5
0.7	0.7	84.7	84.0	0	0.66	75.6
0.8	0.8	88.3	87.6	0	0.74	77.9
0.9	0.9	88.3	87.6	0	0.68	80.0
0.9999	1.0	99.3	89.4	0	9.86	86.5
S003-007	MA 2	MI 3	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	44.1	40.8	02.8	0.50	52.1
0.6	0.6	60.7	57.0	03.0	0.70	54.1
0.7	0.7	63.1	57.0	04.4	0.74	60.8
0.8	0.8	68.5	58.2	09.4	0.88	66.6
0.9	0.9	80.9	58.2	21.6	1.10	79.5
0.9999	1.0	97.2	58.2	29.2	9.78	94.1
S003-007	MA 1	MI 6	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	31.7	27.0	04.4	0.26	26.2
0.6	0.6	40.7	29.4	10.8	0.46	34.5
0.7	0.7	46.7	29.4	16.8	0.48	43.4
0.8	0.8	55.2	29.4	25.0	0.78	57.2
0.9	0.9	86.5	30.0	55.4	1.14	69.8
0.9999	1.0	99.2	30.0	59.4	9.78	83.3
S003-007	MA 0	MI 9	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	18.4	0	18.0	0.44	20.6
0.6	0.6	33.6	0	32.8	0.84	33.5
0.7	0.7	46.6	0	45.4	1.18	47.5
0.8	0.8	73.5	0	71.6	1.90	66.7
0.9	0.9	88.1	0	85.8	2.32	73.3
0.9999	1.0	97.3	0	87.6	9.74	84.6

(MA = Major Gene, MI = Minor Gene, SM = Small Gene, AL = alpha, AD BH = bonferroni-holm adjustment, RP 10 = Ten Recurrent Parents per Replication, h.sq = Heritability, c.to = Correct Detection in Total, c.ma = Correct Detection of Major Genes, c.mi = Correct Detection of Minor Genes, c.sm = Correct Detection of Small Genes, f.po = False Positive). The number next to MA, MI, SM indicates the number of QTL simulated, for example MA3 MI0 10SM means 3 Major Genes of Size 30 each, 0 minor genes of size 10, and 10 small genes of size 1 each.

Table 31: Results of *PopSim* (Linux) Simulation of '*IL-Tests-p10*' with Repetitions Reduced to 50

S003-007	MA 3	MI 0	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	56.1	55.8	0	0.34	47.3
0.6	0.6	78.0	77.4	0	0.56	59.4
0.7	0.7	86.3	85.8	0	0.54	71.2
0.8	0.8	88.8	88.2	0	0.56	73.1
0.9	0.9	88.8	88.2	0	0.58	72.7
0.9999	1.0	98.1	88.2	0	9.86	81.1
S003-007	MA 2	MI 3	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	49.3	46.2	02.8	0.32	35.8
0.6	0.6	61.4	55.2	05.6	0.58	42.5
0.7	0.7	65.8	57.6	07.6	0.64	54.1
0.8	0.8	71.0	58.8	11.4	0.80	62.1
0.9	0.9	83.8	58.8	23.6	1.40	70.3
0.9999	1.0	98.0	58.8	29.4	9.80	83.1
S003-007	MA 1	MI 6	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	35.0	27.0	07.6	0.36	27.6
0.6	0.6	42.7	29.4	12.8	0.54	31.1
0.7	0.7	51.2	28.8	21.6	0.80	34.7
0.8	0.8	64.9	29.4	34.4	1.14	51.9
0.9	0.9	84.7	29.4	53.6	1.68	66.4
0.9999	1.0	98.4	30.0	58.6	9.84	78.2
S003-007	MA 0	MI 9	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	24.1	0	23.8	0.28	15.8
0.6	0.6	34.3	0	33.6	0.68	19.9
0.7	0.7	52.1	0	51.0	1.12	35.9
0.8	0.8	76.7	0	75.0	1.74	53.4
0.9	0.9	89.3	0	87.4	1.90	65.7
0.9999	1.0	98.1	0	88.4	9.74	74.8

(MA = Major Gene, MI = Minor Gene, SM = Small Gene, AL = alpha, AD BH = bonferroni-holm adjustment, RP 10 = Ten Recurrent Parents per Replication, h.sq = Heritability, c.to = Correct Detection in Total, c.ma = Correct Detection of Major Genes, c.mi = Correct Detection of Minor Genes, c.sm = Correct Detection of Small Genes, f.po = False Positive). The number next to MA, MI, SM indicates the number of QTL simulated, for example MA3 MI0 10SM means 3 Major Genes of Size 30 each, 0 minor genes of size 10, and 10 small genes of size 1 each.

Table 32: Results of *PopSim* (Windows) Simulation of '*IL-Tests-p10*' with Repetitions Reduced to 50

S003-007	MA 3	MI 0	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	57.4	57.0	0	0.38	38.6
0.6	0.6	75.5	75.0	0	0.52	64.0
0.7	0.7	86.3	85.8	0	0.52	70.2
0.8	0.8	88.7	88.2	0	0.54	70.7
0.9	0.9	88.8	88.2	0	0.56	73.2
0.9999	1.0	98.1	88.2	0	9.86	81.2
S003-007	MA 2	MI 3	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	50.1	46.8	03.0	0.30	41.8
0.6	0.6	60.6	54.6	05.4	0.56	49.8
0.7	0.7	66.5	58.8	07.0	0.68	54.5
0.8	0.8	71.0	58.8	11.2	1.00	61.4
0.9	0.9	86.0	58.8	25.8	1.44	70.2
0.9999	1.0	98.0	58.8	29.4	9.80	83.3
S003-007	MA 1	MI 6	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	36.3	28.8	07.2	0.30	27.0
0.6	0.6	42.1	28.2	13.4	0.46	32.5
0.7	0.7	51.6	29.4	21.4	0.84	37.8
0.8	0.8	64.0	29.4	33.6	1.02	50.1
0.9	0.9	85.4	29.4	54.4	1.60	67.0
0.9999	1.0	98.4	30.0	58.6	9.84	78.1
S003-007	MA 0	MI 9	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	23.6	0	23.2	0.38	16.2
0.6	0.6	38.3	0	37.6	0.72	25.6
0.7	0.7	52.4	0	51.2	1.16	40.9
0.8	0.8	76.6	0	75.0	1.64	53.7
0.9	0.9	90.1	0	88.0	2.12	66.0
0.9999	1.0	98.1	0	88.4	9.74	74.9

(MA = Major Gene, MI = Minor Gene, SM = Small Gene, AL = alpha, AD BH = bonferroni-holm adjustment, RP 10 = Ten Recurrent Parents per Replication, h.sq = Heritability, c.to = Correct Detection in Total, c.ma = Correct Detection of Major Genes, c.mi = Correct Detection of Minor Genes, c.sm = Correct Detection of Small Genes, f.po = False Positive). The number next to MA, MI, SM indicates the number of QTL simulated, for example MA3 MI0 10SM means 3 Major Genes of Size 30 each, 0 minor genes of size 10, and 10 small genes of size 1 each.

Table 33: Results of *Plabsoft* Simulation of '*IL-Tests-p10*' at Original Repetitions of 5000

S003-007	MA 3	MI 0	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	55.5	55.0	0	0.448	48.4
0.6	0.6	74.8	74.2	0	0.608	62.4
0.7	0.7	86.0	85.3	0	0.715	71.7
0.8	0.8	88.4	87.6	0	0.753	75.2
0.9	0.9	88.6	87.8	0	0.775	77.1
0.9999	1.0	98.0	88.2	0	9.820	84.7
S003-007	MA 2	MI 3	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	48.0	45.4	02.25	0.402	39.0
0.6	0.6	58.7	54.8	03.35	0.514	47.3
0.7	0.7	64.4	58.2	05.55	0.616	52.8
0.8	0.8	70.3	58.7	10.88	0.782	59.0
0.9	0.9	83.2	58.8	23.28	1.112	69.3
0.9999	1.0	98.3	59.0	29.50	9.826	81.7
S003-007	MA 1	MI 6	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	33.7	26.9	06.52	0.336	28.0
0.6	0.6	40.1	28.9	10.78	0.455	33.8
0.7	0.7	48.2	29.3	18.28	0.667	40.7
0.8	0.8	63.1	29.3	32.70	1.059	53.2
0.9	0.9	85.2	29.4	54.30	1.543	68.8
0.9999	1.0	98.3	29.5	58.97	9.836	81.0
S003-007	MA 0	MI 9	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	19.7	0	19.2	0.410	16.9
0.6	0.6	31.7	0	31.0	0.705	27.0
0.7	0.7	49.7	0	48.5	1.161	40.4
0.8	0.8	75.1	0	73.3	1.748	58.4
0.9	0.9	89.6	0	87.6	2.058	69.0
0.9999	1.0	98.3	0	88.5	9.835	78.1

(MA = Major Gene, MI = Minor Gene, SM = Small Gene, AL = alpha, AD BH = bonferroni-holm adjustment, h.sq = Heritability, RP 10 = Ten Recurrent Parents per Replication, c.to = Correct Detection in Total, c.ma = Correct Detection of Major Genes, c.mi = Correct Detection of Minor Genes, c.sm = Correct Detection of Small Genes, f.po = False Positive). The number next to MA, MI, SM indicates the number of QTL simulated, for example MA3 MI0 10SM means 3 Major Genes of Size 30 each, 0 minor genes of size 10, and 10 small genes of size 1 each.

Table 34: Results of *PopSim* Simulation of '*IL-Tests-p10*' at Original Repetitions of 5000

S003-007	MA 3	MI 0	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	54.7	54.3	0	0.439	47.8
0.6	0.6	75.0	74.4	0	0.591	62.2
0.7	0.7	86.1	85.4	0	0.697	71.0
0.8	0.8	88.5	87.8	0	0.735	74.8
0.9	0.9	88.8	88.0	0	0.757	76.9
0.9999	1.0	98.3	88.5	0	9.827	84.6
S003-007	MA 2	MI 3	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	48.4	45.7	02.27	0.391	40.0
0.6	0.6	58.5	54.7	03.32	0.501	48.1
0.7	0.7	64.1	58.1	05.45	0.603	53.3
0.8	0.8	70.1	58.5	10.76	0.779	59.1
0.9	0.9	83.3	58.6	23.58	1.100	69.2
0.9999	1.0	98.1	58.8	29.48	9.823	81.7
S003-007	MA 1	MI 6	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	33.8	26.8	06.62	0.349	27.9
0.6	0.6	40.5	28.8	11.25	0.481	33.7
0.7	0.7	48.3	29.2	18.38	0.685	40.5
0.8	0.8	63.0	29.2	32.65	1.075	52.5
0.9	0.9	84.9	29.3	54.09	1.567	68.2
0.9999	1.0	98.2	29.5	58.89	9.832	80.2
S003-007	MA 0	MI 9	SM 10	AL 0.05	AD BH	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	19.5	0	19.1	0.401	17.3
0.6	0.6	32.1	0	31.4	0.716	26.8
0.7	0.7	50.8	0	49.6	1.189	40.5
0.8	0.8	75.2	0	73.5	1.748	58.4
0.9	0.9	89.6	0	87.5	2.078	69.4
0.9999	1.0	98.4	0	88.6	9.840	78.5

(MA = Major Gene, MI = Minor Gene, SM = Small Gene, AL = alpha, AD BH = bonferroni-holm adjustment, h.sq = Heritability, RP 10 = Ten Recurrent Parents per Replication, c.to = Correct Detection in Total, c.ma = Correct Detection of Major Genes, c.mi = Correct Detection of Minor Genes, c.sm = Correct Detection of Small Genes, f.po = False Positive). The number next to MA, MI, SM indicates the number of QTL simulated, for example MA3 MI0 10SM means 3 Major Genes of Size 30 each, 0 minor genes of size 10, and 10 small genes of size 1 each.

Table 35: Results of *Plabsoft* Simulation of '*Overlap-DTrun-MSeg-np*' with Repetitions Reduced to 50

S003-009	MA 3	MI 0	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	30.1	30.0	0	0.14	46.1
0.6	0.6	56.7	56.4	0	0.34	41.0
0.7	0.7	81.9	81.6	0	0.34	28.5
0.8	0.8	89.8	89.4	0	0.38	26.8
0.9	0.9	90.4	90.0	0	0.42	24.7
0.9999	1.0	95.6	90.0	0	5.56	18.7
S003-009	MA 2	MI 3	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	31.8	30.0	01.6	0.20	14.8
0.6	0.6	48.5	45.6	02.6	0.28	33.7
0.7	0.7	60.3	56.4	03.4	0.54	32.3
0.8	0.8	66.1	60.0	05.6	0.50	25.1
0.9	0.9	78.6	60.0	17.8	0.82	27.0
0.9999	1.0	96.2	60.0	30.0	6.24	18.1
S003-009	MA 1	MI 6	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	28.8	24.6	04.0	0.22	20.9
0.6	0.6	38.1	29.4	08.4	0.32	35.2
0.7	0.7	39.2	29.4	09.4	0.38	13.2
0.8	0.8	52.4	30.0	21.8	0.64	19.5
0.9	0.9	72.3	30.0	41.4	0.94	22.6
0.9999	1.0	98.2	30.0	60.0	8.22	15.4
S003-009	MA 0	MI 9	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	12.9	0	12.6	0.26	16.12
0.6	0.6	13.3	0	13.0	0.26	09.41
0.7	0.7	31.3	0	30.6	0.68	15.88
0.8	0.8	57.2	0	56.2	0.98	13.66
0.9	0.9	87.3	0	85.8	1.52	13.22
0.9999	1.0	99.8	0	89.8	9.96	13.58

(MA = Major Gene, MI = Minor Gene, SM = Small Gene, AL = alpha, AD Dunnett = Dunnett test, RP 10 = Ten Recurrent Parents per Replication, h.sq = Heritability, c.to = Correct Detection in Total, c.ma = Correct Detection of Major Genes, c.mi = Correct Detection of Minor Genes, c.sm = Correct Detection of Small Genes, f.po = False Positive). The number next to MA, MI, SM indicates the number of QTL simulated, for example MA3 MI0 10SM means 3 Major Genes of Size 30 each, 0 minor genes of size 10, and 10 small genes of size 1 each.

Table 36: Results of *PopSim* (Linux) Simulation of '*Overlap-DTrun-MSeg-np*' with Repetitions Reduced to 50

S003-009	MA 3	MI 0	SM 10	AL 0.05	Ad Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	41.7	41.4	0	0.32	63.0
0.6	0.6	58.5	58.2	0	0.26	55.3
0.7	0.7	71.7	71.4	0	0.30	47.3
0.8	0.8	88.7	88.2	0	0.46	39.1
0.9	0.9	90.4	90.0	0	0.42	31.1
0.9999	1.0	94.9	90.0	0	4.86	22.0
S003-009	MA 2	MI 3	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	33.2	30.6	02.4	0.20	22.2
0.6	0.6	51.2	46.8	04.0	0.44	61.1
0.7	0.7	60.9	57.0	03.6	0.32	38.4
0.8	0.8	67.5	60.0	07.0	0.50	24.3
0.9	0.9	77.8	60.0	17.0	0.78	21.5
0.9999	1.0	97.0	60.0	30.0	6.96	16.5
S003-009	MA 1	MI 6	SM 10	AL 0.05	Ad Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	28.1	24.0	04.0	0.10	23.7
0.6	0.6	35.9	28.2	07.4	0.26	17.0
0.7	0.7	43.6	30.0	13.2	0.38	14.7
0.8	0.8	50.8	30.0	20.2	0.58	21.0
0.9	0.9	73.0	30.0	42.2	0.80	21.2
0.9999	1.0	98.6	30.0	60.0	8.64	17.4
S003-009	MA 0	MI 9	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	09.54	0	09.4	0.14	19.3
0.6	0.6	20.44	0	20.2	0.24	15.0
0.7	0.7	27.08	0	26.8	0.28	12.6
0.8	0.8	55.16	0	54.4	0.76	17.1
0.9	0.9	83.78	0	82.6	1.18	22.8
0.9999	1.0	99.48	0	89.6	9.88	15.0

(MA = Major Gene, MI = Minor Gene, SM = Small Gene, AL = alpha, AD Dunnett = Dunnett test, RP 10 = Ten Recurrent Parents per Replication, h.sq = Heritability, c.to = Correct Detection in Total, c.ma = Correct Detection of Major Genes, c.mi = Correct Detection of Minor Genes, c.sm = Correct Detection of Small Genes, f.po = False Positive). The number next to MA, MI, SM indicates the number of QTL simulated, for example MA3 MI0 10SM means 3 Major Genes of Size 30 each, 0 minor genes of size 10, and 10 small genes of size 1 each.

Table 37: Results of *PopSim* (Windows) Simulation of '*Overlap-DTrun-MSeg-np*' with Reduced Repetitions

S003-009	MA 3	MI 0	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	32.0	31.8	0	0.16	30.6
0.6	0.6	53.7	53.4	0	0.26	45.6
0.7	0.7	84.4	84.0	0	0.36	34.0
0.8	0.8	90.4	90.0	0	0.40	25.7
0.9	0.9	90.5	90.0	0	0.52	29.8
0.9999	1.0	95.0	90.0	0	5.00	22.1
S003-009	MA 2	MI 3	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	33.9	31.8	02.0	0.10	51.9
0.6	0.6	43.0	40.2	02.6	0.24	32.1
0.7	0.7	62.0	57.0	04.6	0.38	32.8
0.8	0.8	69.8	60.0	09.2	0.56	28.0
0.9	0.9	77.0	60.0	16.2	0.80	26.9
0.9999	1.0	96.9	60.0	30.0	6.94	16.5
S003-009	MA 1	MI 6	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	27.5	23.4	04.0	0.14	23.5
0.6	0.6	35.5	28.2	07.0	0.30	32.9
0.7	0.7	43.6	29.4	13.8	0.44	25.5
0.8	0.8	50.3	30.0	19.8	0.50	20.5
0.9	0.9	73.0	30.0	42.2	0.80	19.3
0.9999	1.0	98.4	30.0	60.0	8.38	17.4
S003-009	MA 0	MI 9	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	11.3	0	11.2	0.12	20.9
0.6	0.6	22.8	0	22.6	0.24	16.0
0.7	0.7	38.9	0	38.4	0.52	16.3
0.8	0.8	60.2	0	59.4	0.84	14.8
0.9	0.9	86.0	0	84.8	1.22	23.7
0.9999	1.0	99.5	0	89.6	9.92	15.0

(MA = Major Gene, MI = Minor Gene, SM = Small Gene, AL = alpha, AD Dunnett = Dunnett test, RP 10 = Ten Recurrent Parents per Replication, h.sq = Heritability, c.to = Correct Detection in Total, c.ma = Correct Detection of Major Genes, c.mi = Correct Detection of Minor Genes, c.sm = Correct Detection of Small Genes, f.po = False Positive). The number next to MA, MI, SM indicates the number of QTL simulated, for example MA3 MI0 10SM means 3 Major Genes of Size 30 each, 0 minor genes of size 10, and 10 small genes of size 1 each.

Table 38: Results of *Plabsoft* Simulation of '*Overlap-DTrun-MSeg-np*' at Original Repetitions of 5000

S003-009	MA 3	MI 0	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	36.5	36.3	0	0.209	41.8
0.6	0.6	57.6	57.3	0	0.328	36.6
0.7	0.7	78.9	78.4	0	0.430	34.6
0.8	0.8	88.9	88.4	0	0.492	32.4
0.9	0.9	90.2	89.7	0	0.510	27.9
0.9999	1.0	95.5	89.8	0	5.734	18.4
S003-009	MA 2	MI 3	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	34.7	32.9	01.56	0.202	34.7
0.6	0.6	49.4	46.8	02.25	0.291	33.1
0.7	0.7	60.7	56.6	03.75	0.371	30.7
0.8	0.8	66.8	59.7	06.64	0.445	25.4
0.9	0.9	77.7	59.9	17.22	0.644	24.2
0.9999	1.0	96.7	59.9	29.94	6.834	18.1
S003-009	MA 1	MI 6	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	26.9	22.7	04.04	0.188	26.3
0.6	0.6	34.9	27.8	06.83	0.271	24.3
0.7	0.7	41.8	29.7	11.75	0.366	22.7
0.8	0.8	52.4	29.9	21.91	0.580	22.2
0.9	0.9	77.3	29.9	46.42	0.983	21.1
0.9999	1.0	98.1	29.9	59.87	8.270	17.0
S003-009	MA 0	MI 9	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	12.0	0	11.8	0.196	19.7
0.6	0.6	20.2	0	19.9	0.332	18.7
0.7	0.7	34.4	0	33.8	0.588	18.8
0.8	0.8	57.4	0	56.4	0.969	19.2
0.9	0.9	86.9	0	85.6	1.395	19.7
0.9999	1.0	99.7	0	89.9	9.813	16.5

(MA = Major Gene, MI = Minor Gene, SM = Small Gene, AL = alpha, AD Dunnett = Dunnett test, RP 10 = Ten Recurrent Parents per Replication, h.sq = Heritability, c.to = Correct Detection in Total, c.ma = Correct Detection of Major Genes, c.mi = Correct Detection of Minor Genes, c.sm = Correct Detection of Small Genes, f.po = False Positive). The number next to MA, MI, SM indicates the number of QTL simulated, for example MA3 MI0 10SM means 3 Major Genes of Size 30 each, 0 minor genes of size 10, and 10 small genes of size 1 each.

Table 39: Results of *PopSim* Simulation of '*Overlap-DTrun-MSeg-np*' at Original Repetitions of 5000

S003-009	MA 3	MI 0	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	36.5	36.3	0	0.198	41.3
0.6	0.6	57.9	57.6	0	0.314	38.0
0.7	0.7	78.5	78.1	0	0.415	35.0
0.8	0.8	88.9	88.4	0	0.472	32.4
0.9	0.9	90.2	89.8	0	0.482	26.6
0.9999	1.0	95.6	89.8	0	5.886	17.7
S003-009	MA 2	MI 3	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	34.2	32.5	01.52	0.202	34.0
0.6	0.6	49.8	47.2	02.26	0.289	33.2
0.7	0.7	60.4	56.6	03.35	0.365	29.5
0.8	0.8	66.9	49.8	06.69	0.450	26.3
0.9	0.9	77.9	59.9	17.32	0.666	23.2
0.9999	1.0	96.7	59.9	29.93	6.924	17.8
S003-009	MA 1	MI 6	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	27.1	22.9	03.95	0.201	26.6
0.6	0.6	34.8	27.9	06.68	0.271	25.1
0.7	0.7	42.0	29.8	11.81	0.390	23.0
0.8	0.8	52.3	30.0	21.76	0.584	22.0
0.9	0.9	77.0	30.0	46.01	0.993	21.9
0.9999	1.0	98.1	30.0	59.87	8.286	17.1
S003-009	MA 0	MI 9	SM 10	AL 0.05	AD Dunnett	RP 10
	h.sq	c.to	c.ma	c.mi	c.sm	f.po
0.5	0.5	12.3	0	12.1	0.210	20.4
0.6	0.6	20.7	0	20.4	0.350	19.5
0.7	0.7	34.1	0	33.5	0.583	18.0
0.8	0.8	57.7	0	56.7	1.011	19.1
0.9	0.9	87.1	0	85.7	1.432	19.4
0.9999	1.0	99.6	0	89.8	9.818	16.5

(MA = Major Gene, MI = Minor Gene, SM = Small Gene, AL = alpha, AD Dunnett = Dunnett test, RP 10 = Ten Recurrent Parents per Replication, h.sq = Heritability, c.to = Correct Detection in Total, c.ma = Correct Detection of Major Genes, c.mi = Correct Detection of Minor Genes, c.sm = Correct Detection of Small Genes, f.po = False Positive). The number next to MA, MI, SM indicates the number of QTL simulated, for example MA3 MI0 10SM means 3 Major Genes of Size 30 each, 0 minor genes of size 10, and 10 small genes of size 1 each.

The Table 38 presents the outcome of the 'Overlap-DTrun-MSeg-np' simulation performed by the *Plabsoft* where as the Table 39 shows the output of *PopSim* for the same simulation.

3.6 **Simulation Time Comparison of Marker-Assisted Backcrossing Example**

The Marker-Assisted Backcrossing examples utilized the parallelized part of *PopSim* thus a substantial improvement in speed was expected in case of these simulations

The examples *2cM Equally* and *2cM Randomly* were simulated thrice at reduced repetitions of 100 instead of the original 10000. Along with these simulations, the timing of all the simulations was documented for both *Plabsoft* and *PopSim* by running them at the original 10000 repetitions.

3.6.1 2cM Equally with Repetitions reduced to 100

Reduced repetitions of 100 in lieu of the original 10000 were used for the *2cM Equally* simulation. The time required for the execution of this simulation was noted for both *PopSim* and *Plabsoft*. After comparing the timing of the three runs, it was noted that on average *PopSim* was taking only 40% of the time that *Plabsoft* was using to execute this simulation, which means that *PopSim* was providing a time savings of 29.09694 Seconds (60%).

The time recordings from all of the three runs of *2cM Equally* simulation on both *PopSim* and *Plabsoft* are displayed in the Table 40.

3.6.2 2cM Equally at Original Repetitions

The time required by *PopSim* and *Plabsoft* to execute the *2cM Equally* simulation at its original repetitions of 10000 was also recorded once. For this scenario, *PopSim* required only 32.4568 Min (0.5409467 Hours) to complete the simulation compared to the 1.35314 Hours taken by *Plabsoft*. That is a time savings of 48.7316 minutes (60%).

Table 40: Comparison of Time of 2cM Equally

at reduced Repetitions in seconds

Run	*Plabsoft*	*PopSim*
1	49.45191	20.35669
2	48.70195	19.82495
3	48.854	19.53539
Average	49.00262	19.90568

3.6.3 2cM Randomly with Repetitions reduced to 100

For this case, the *2cM Randomly* simulation was performed alternatively for three times by both *PopSim* and *Plabsoft* at a reduced repetitions of 100 in place of the original 10000. On examining the timings, it was observed that on average *PopSim* was taking 29.1209966 Seconds (60%) less time to finish the simulation than *Plabsoft*.

The Table 41 illustrates the time comparison of each run of the simulation done on both the software.

3.6.4 2cM Randomly at Original Repetitions

The *2cM Randomly* simulation was also performed at the original repetitions of 10000 by both software where *PopSim* only need 32.1738 minutes (0.53623 Hours) to complete the simulation instead of the 1.353707 Hours taken by *Plabsoft*. To simplify the results, *PopSim* required 49.04862 minutes (60%) less than *Plabsoft* to finish this simulation.

Table 41: Comparison of Time of 2cM Randomly

at reduced Repetitions in seconds

Run	*Plabsoft*	*PopSim*
1	48.98091	19.58294
2	48.60107	19.56737
3	48.5717	19.64038
Average	48.7178933	19.5968967

3.6.5 5cM Equally at Original Repetitions

PopSim was able to finish the *5cM Equally* simulation running it at its original 10000 repetitions in only 21.15053 minutes where as *Plabsoft* required 46.06902 minutes to accomplish the same. Practically meaning that by using *PopSim* 24.91849 Minutes (54%) of the system time was saved over *Plabsoft*.

3.6.6 5cM Randomly at Original Repetitions

The use of *PopSim* saved 25.00118 minutes (54%) of user time by completing the *5cM Randomly* simulation set at its original repetitions of 10000 in only 21.09401 minutes in lieu of the 46.09519 minutes taken by *Plabsoft* to finish the same simulation.

3.6.7 10cM Equally at Original Repetitions

The repetitions of the *10cM Equally* simulation were set to the original 10000 for testing it on both the *Plabsoft* and *PopSim*. To complete this simulation *Plabsoft* required 34.47579 minutes of the system time whereas the *PopSim* was finished with the same simulation in only 17.18703 minutes. Thus, the use of *PopSim* gave a time savings of 17.28876 minutes (approx 50%) over *Plabsoft*.

3.6.8 10cM Randomly at Original Repetitions

A time savings of 17.10219 minutes (approx. 50%) was experienced by *PopSim* over *Plabsoft* for the *10cM Randomly* simulation executed at its original repetitions of 10000. This simulation finished in 34.38053 minutes by *Plabsoft* while *PopSim* was able to achieve the same in only 17.10219 minutes.

3.6.9 20cM Equally at Original Repetitions

On simulating the *20cM Equally* simulation with repetitions set to original 10000 it was observed that *PopSim* was able to finish it in only 15.28368 Min while *Plabsoft* required an extra 13.22299 minutes to complete the simulation in 28.50667 minutes. Therefore, to simplify the findings it can be seen that *PopSim* needed approximately 46% less time than *Plabsoft* to execute the *20cM Equally* simulation.

3.6.10 20cM Randomly at Original Repetitions

Both *PopSim* and *Plabsoft* executed the *20cM Randomly* simulation while the repetitions were kept unchanged at its original 10000. *Plabsoft* was able to complete this simulation in 28.55255 minutes while *PopSim* was done with it in mere 15.22368 minutes, thus saving 13.32887 minutes (47%) of user time.

Finally, it can be concluded that as far as the simulations that exploit the parallelized part of *PopSim* are concerned a substantial improvement in speed is gained over *Plabsoft* by performing the simulation in *PopSim*.

3.7 **Simulation Time Comparison of QTL Mapping Example**

The QTL Mapping examples did not utilize the parallelized part of *PopSim*. The *IL-Tests-p10* and *Overlap-DTrun-MSeg-np* were timed in both *Plabsoft* and *PopSim*. Since these examples did not use the parallelized part of *PopSim* or use the paralleled part of *PopSim* in negligible part of the simulation no time improvement was expected for these simulations.

The examples were executed thrice at reduced repetitions of 50 instead of the original 5000 and once at 5000 repetitions on both *Plabsoft* and *PopSim* on the Linux System.

3.7.1 IL-Tests-p10 with Repetitions reduced to 50

For this setting, the *IL-Tests-p10* simulation was executed with its repetitions reduced to 50 from 5000. This simulation was performed three times by both *PopSim* and *Plabsoft*. It was observed that *PopSim* was not significantly faster than *Plabsoft* taking 1.4 seconds (1.5%) less.

The measurements from the three runs of the simulation on both the *Plabsoft* and *PopSim* are exhibited in the Table 42.

3.7.2 IL-Tests-p10 at Original Repetitions

The second measurements made on the *IL-Tests-p10* simulation were prepared at the original repetitions of 5000 once on both *Plabsoft* and *PopSim*. To execute this simulation

PopSim required 8.933733 Hours in lieu of the 9.093236 Hours used by *Plabsoft* that is a negligible time saving of 9.57 minutes (1.75%).

3.7.3 Overlap-DTrun-MSeg-np with Repetitions reduced to 50

The repetitions of the *Overlap-DTrun-MSeg-np* simulation were reduced to 50 from 5000 for this measurement and it was simulated on both software three times. On examining the timings it was noted that on average *PopSim* was taking 1.299 Seconds (2%) less than *Plabsoft*.

The Table 43 shows the recorded time in minutes from the three runs of the simulation on both the *PopSim* and *Plabsoft*.

3.7.4 Overlap-DTrun-MSeg-np at Original Repetitions

The time taken by the *Overlap-DTrun-MSeg-np* simulation was measured once again on both *PopSim* and *Plabsoft* but this time at its original 5000 repetitions. It was discovered that *PopSim* was taking a meager 4.25% less time (21.23 minutes) by completing the simulation in 7.980589 Hours in comparison to the 8.334438 Hours used by *Plabsoft*.

Table 42: Comparison of Time of IL-Tests-p10
at reduced Repetitions in minutes

Run	*Plabsoft*	*PopSim*
1	1.485742	1.50183
2	1.514941	1.477496
3	1.518744	1.473896
Average	1.507743	1.4844073

Table 43: Comparison of Time of Overlap-DTrun-MSeg-np
at reduced Repetitions in minutes

Run	*Plabsoft*	*PopSim*
1	1.061712	1.030557
2	1.056964	1.044703
3	1.063997	1.042463
Average	1.060891	1.039241

In the end, it can be concluded that as far as those simulations are concerned that do not utilize the parallelized part of *PopSim*, the improvement in speed acquired by using *PopSim* over *Plabsoft* is insignificant.

3.8 Compilation of PopSim

Another requirement of the project was that *PopSim* should compile cleanly under both Windows and Linux operating systems. All of the warning and error messages were solved before the actual testing of the software was started. The current version of *PopSim* compiles cleanly regardless of which of the two operating systems it is compiled on.

3.8.1 Compilation of the C code

It was mandatory that the C code of *PopSim* should compile cleanly without producing any errors or warnings. The compilation of the C code of *PopSim* was also done separately to check if it was giving any errors or warnings.

The output given out by GCC on compiling the ‘c’ file of *PopSim* on the Linux System is shown in Figure 1 while the compiling outcome of the ‘c’ file on the Windows System is displayed in Figure 2.

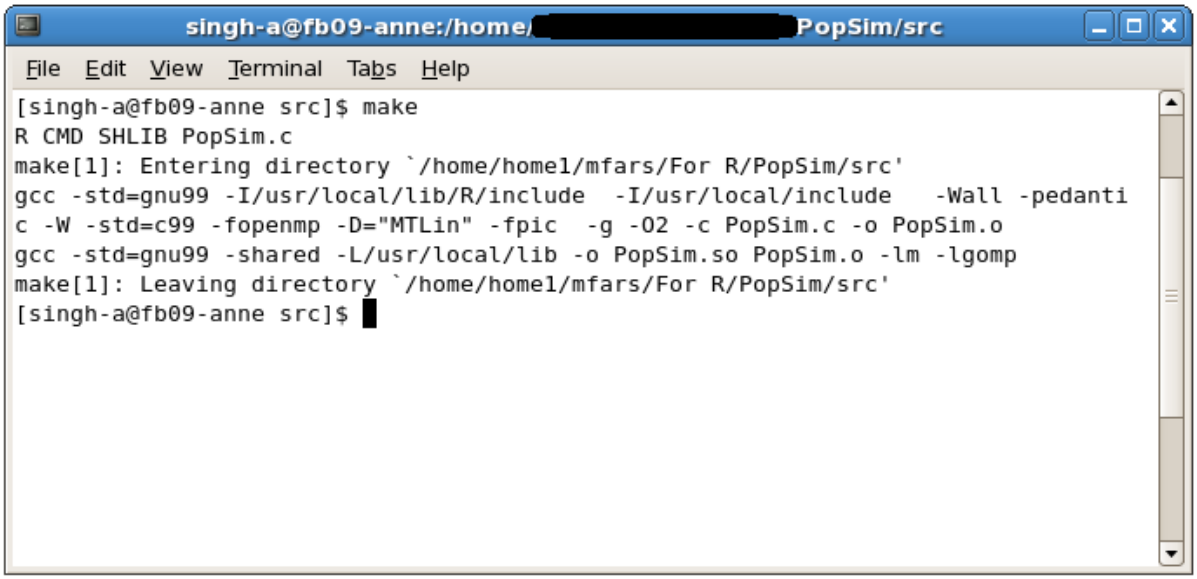


Figure 1: Compilation results of C code under Linux

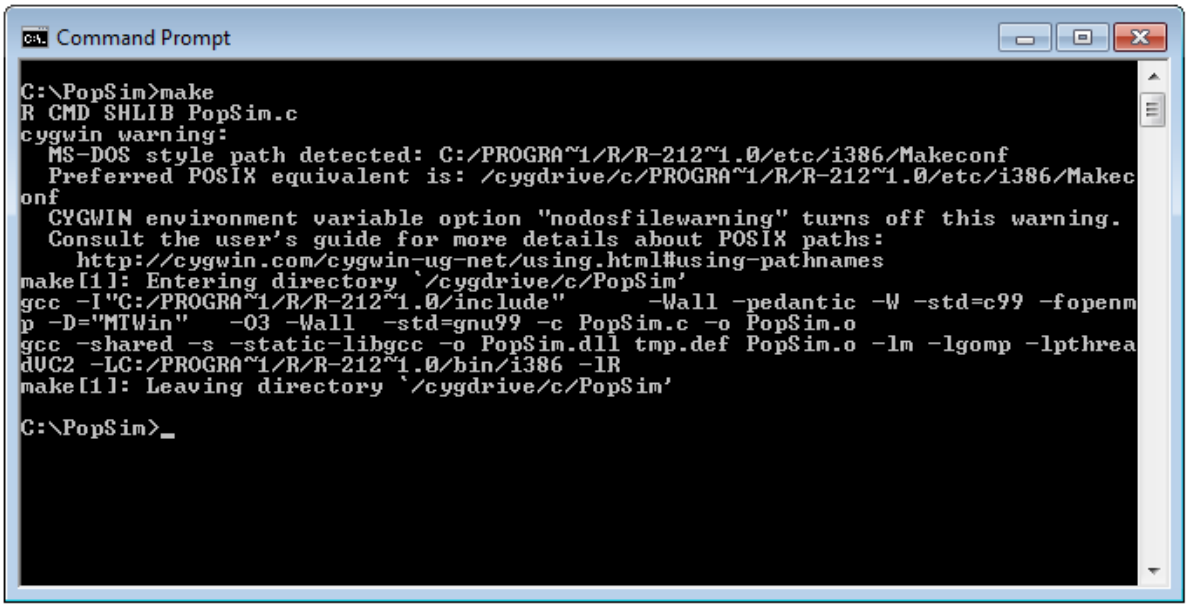


Figure 2: Compilation results of C code under Windows

It might be noted that some warnings can be seen in the compiler output from the Windows system but these warnings are not caused by *PopSim*. These warnings are actually caused by the compiler itself and have nothing to do with the *PopSim* code, which can be verified by actually reading the warning message in the Figure 2.

3.8.2 Creation of the *PopSim* R-Package:

Cleanly compiling the C code of *PopSim* would be useless if *PopSim* did not compiled cleanly while its package was created using the 'R CMD build' command. Not only does the 'c' file of the current version of *PopSim* compiles cleanly but it also compiles cleanly without throwing any warnings or errors.

The results of the creation of the *PopSim* package by executing the 'R CMD build' command on the Windows system are exhibited in the Figure 3.

Here again, it could be observed that some warnings are showing up in the output from the Windows system. The generation of these warning messages are not caused by errors in the *PopSim* code, but are caused by the compiler itself because of the format of system path. This can be verified by looking at the warning message in the Figure 3.

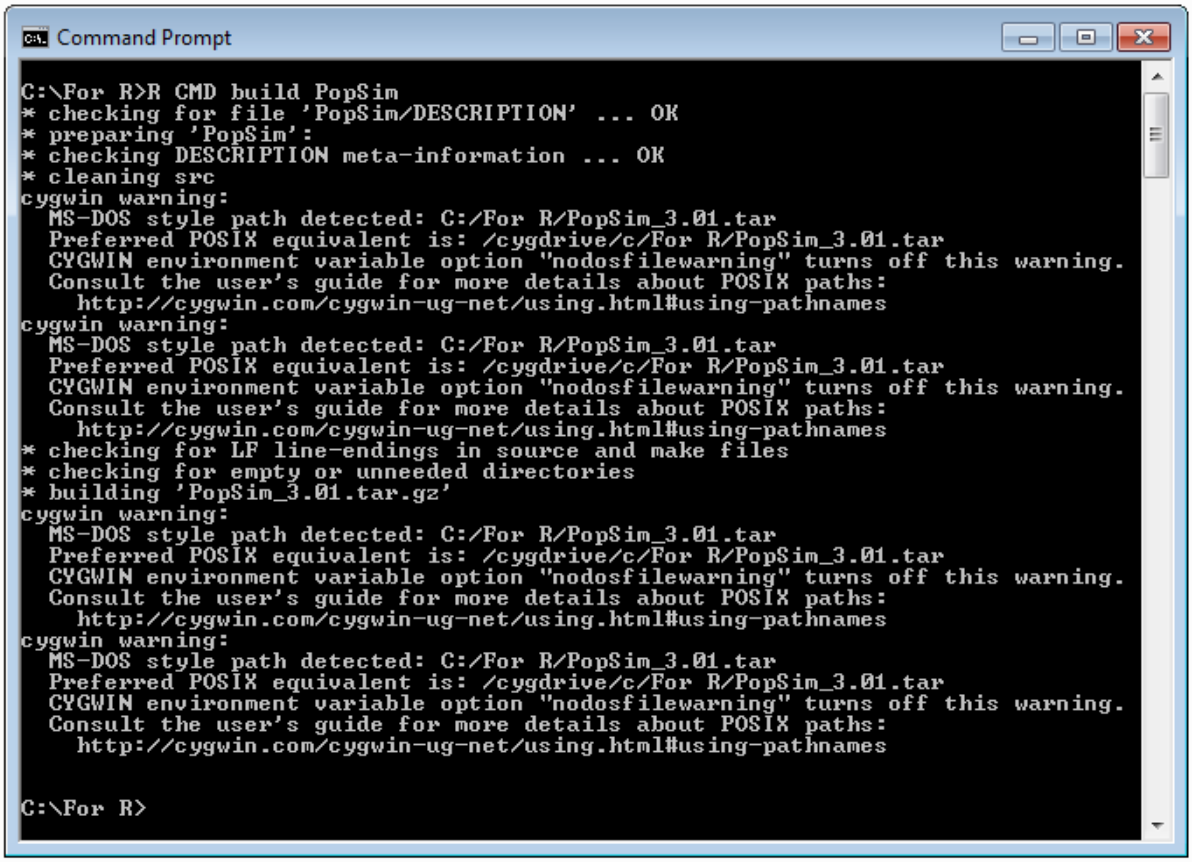


Figure 3: Package creation under Windows

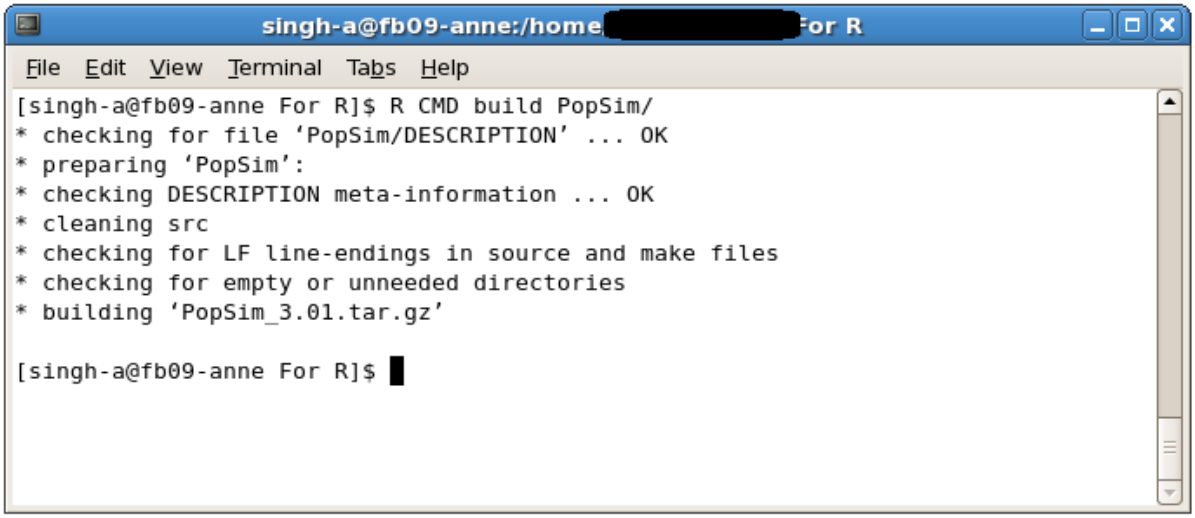


Figure 4: Package creation under Linux

The Figure 4 displays the outcome of creating the *PopSim* package on the Linux system using the 'R CMD build' command.

3.9 Summary of Results:

To summarize, it was observed that *PopSim* produced results, which were comparable to the results produced by *Plabsoft* for the artificial example and for both the marker-assisted backcrossing and QTL mapping examples as well. In the artificial example, it was noted that on increasing the size of the population the time difference between the execution of the simulation by *PopSim* and *Plabsoft* also increased for the reason that by increasing the population, the percent of the code which can be parallelized increases. On the other hand an increase in the number of repetitions in the artificial example displayed a time difference decrease because *PopSim* can only parallelize one repetition at a time, as the loop running the repetitions is a sequential piece of code run by the parent script. The improvement in time provided by *PopSim* was insignificant over *Plabsoft* in the case of the QTL mapping examples since these did not employ the parallelized part of *PopSim*. Significant improvement of speed was observed for the marker-assisted backcrossing examples as these engaged the parallelized components of *PopSim*. Here *PopSim* was able to give an average savings of 53% with highest time savings of 60% and the lowest time savings of 46% over *Plabsoft*. In addition, it was also confirmed that *PopSim* compiles cleanly under both Linux and Windows without giving any errors or warnings.

4. Discussion

The main objective of the project was to create a multi-threaded program based on *Plabsoft* which can maximize the utilization of the processing power provided by the modern day computers incorporating multi-core CPUs. The second objective was that the new software should not be dependent on third party libraries. It was also essential for the new program to have the capability to be compiled and executed on both the Linux and Windows operating systems. Additionally it was expected that the new program should be able to compile cleanly on both Linux and Windows systems without displaying any errors or warning messages at compile time.

In the beginning, the entire codebase of *Plabsoft* was studied to understand the internal working of the program. The third party libraries on which *Plabsoft* was dependent were identified while examining the internal working of *Plabsoft*. Along with these libraries, the components of *Plabsoft* that were dependent on these libraries were also identified.

It was discovered that *Plabsoft* was chiefly reliant on three external libraries i.e. *GSL, GSL-CBLAS & GMP*. These libraries were used by *Plabsoft* for the generation of high quality random numbers and Poisson distributions. For the new software to be independent of these libraries, alternative approaches that did not use these libraries for the generation of high precision random number and Poisson distributions were needed.

Two different approaches for the generation of high precision random number and Poisson distributions were considered. The first approach involved the designing of a new algorithm from scratch and then implementing it in the new software. The other option was to explore the already existing algorithms and investigate their feasibility for implementation in the new software.

On judging the available options, it was evident that designing a completely new algorithm to generate superior quality random numbers and Poisson distributions will require a lot of time. Furthermore, the new algorithm will necessitate an extra set of tests to

analyze the proper working of the algorithm. Therefore, it was decided it would be better to first try to seek out already existing algorithms and test their feasibility for the new software and if no feasible solution is found only then to proceed with the designing of the new algorithm. Some suitable algorithms were found in *Numerical Recipes (Press, Teukolsky, Vetterling and Flannery 2007)*.

It was essential for the new software to employ the original algorithms used in *Plabsoft* whenever possible with minimal changes to them. New algorithms were designed using the algorithms from the *Numerical Recipes* as reference such that externally they mimicked the functioning of the random numbers and Poisson distribution generator in the GSL library. The source code of the GSL library was thoroughly studied to understand the required framework to create the new algorithms. The new portion of code designed to perform the random number generations and production of Poisson distributions was implemented in *Plabsoft* after making the necessary changes to the source code of *Plabsoft*.

This new version of *Plabsoft* that was using the custom random number and Poisson distribution generator was tested against the original version of *Plabsoft*, which used the previous external libraries for the production of Poisson distributions and random numbers. The testing was carried out using an artificial example simulation. The artificial example was run on both software using different sizes of populations and repetitions. It was observed that the simulation results obtained from both the software were identical irrespective of the population size or the number of repetitions used denoting that the new random number and Poisson distribution algorithm was operating suitably.

After removal of dependencies on external third party libraries, the various strategies supported by GCC for the implementation of multithreading were investigated. Since the new program was going to be an R package consisting of both an R and a C component, it was initially considered to parallelize both the R and C parts. After further exploration, the parallelization of the R part of the program was dropped because the parallelization support in R was still in experimental stages at that time. Therefore, the different models of parallelization available for the C language were considered.

Initially two Parallel Programming models provided by the C language were considered, namely:

1. *Directives-based parallel programming*: This approach uses directives to tell the processor how to distribute data and work across the different processors or cores. This model of parallel programming also provides the facility of compiling the code on a compiler that does not support multithreading in which case the directives appear as comments to the compiler. This method is mainly implemented on shared memory architectures, i.e. multiple processors or multiple cores on a single processor sharing the same memory.
2. *Message-passing parallel programming*: The processors communicate and transfer data by sending and receiving messages in case of this approach. Each processor or core has its own independent local variable that is not accessible to other processors or cores directly. The advantage of this model is that it can be used on both shared memory architectures and distributed memory architectures. On the other hand, the downside of using this method is that it is not possible to compile this code on a compiler which does not support multithreading.

Finally, it was decided to proceed with the directives-based parallel programming model for the new software because of the fact that directives-based parallel programming was more appropriate model for cases using shared memory architecture and the new program was going to be designed for running on machines using multi-core processor sharing the same memory. Moreover, the requirement of the project that the new software should still be capable of compiling on older compilers that did not support multithreading was better handled by the directives-based parallel programming approach.

For the implementation of multithreading in the new program the main methods that were taken into account were *OpenMP*, *MPI* (*Gropp, Lusk and Skjellum 1994*) and *TBB* (*Reinders 2007*). In the end, it was decided to proceed with *OpenMP* for the implementation of multithreading capabilities in the new software. The reason for choosing *OpenMP* over *MPI* and *TBB* was that this approach was better suited to the project. The advantages of using *OpenMP* over the other two approaches in case of the current project are discussed below:

Advantages of *OpenMP* over *MPI* (Message Passing Interface):

1. *OpenMP* supports gradual parallelization, i.e. directives can be added incrementally to the code. This gives the option of adding parallelism to the program in smaller parts.
2. The program designed using *OpenMP* can executed on systems not supporting parallelism without requiring any changes to the code.
3. *OpenMP* is easier to program than *MPI*.
4. Debugging of programs implementing parallelism with *OpenMP* is easier compared to a program utilizing *MPI*.
5. For a person who was not part of the original coding team, the *OpenMP* code is generally easier to understand than *MPI*.
6. The maintenance of software designed by incorporating *OpenMP* code is easier than the equivalent *MPI* version.
7. The *OpenMP* code can be compiled into a serial code by a compiler not supporting *OpenMP* whereas the *MPI* code cannot be compiled by a compiler not supporting *MPI*.
8. The *OpenMP* approach requires less programming changes in the serial code to make the parallel version of the code.

Advantages of *OpenMP* over *TBB* (Intel® Threading Building Blocks):

1. *OpenMP* has less overhead in loops than *TBB*.
2. *OpenMP* is an open standard where as *TBB* is not.
3. With *OpenMP* parallelism can be obtained incrementally.
4. *OpenMP* is simpler.
5. *OpenMP* fits better into the structured coding style of C language than *TBB*, which is designed for highly object-oriented code.
6. *OpenMP* is better suited for large and predictable data parallel problems.

Once it was decided to use *OpenMP* for the implementation of parallelization in the new program, the parts of the program where multiple threading was possible were noted, e.g. crossing, population evaluation and genotype population. *Appendix IV* displays a flowchart

for the crossing algorithm, and a flowchart of the population evaluation algorithm is shown in *Appendix V*. The flow chart of the genotype population algorithm is present in the *Appendix VI*. After this, new algorithms supporting multithreading were designed to replace these parts and coding of the new software *PopSim* was started.

On completion of the coding phase, all of the errors and warnings displayed by *PopSim* were dealt with. This cleanly compiling copy of *PopSim* was tested using the artificial example simulation to remove any runtime problems.

The *PopSim* code was then ported to the Windows version and sections of the code that required any change were marked. Necessary changes were made to these parts of the code so that *PopSim* could be compiled on both the Linux and Windows machines without the need to make any changes to the code.

The final version of *PopSim* is capable of compiling and executing on both the Linux and Windows systems with the requirement of only a slight change in the *Makevars* file between the two versions. Additionally, the Windows version also required the installation of *Pthreads-w32* to add the support for multithreading.

After the coding phase of *PopSim* was completed, the testing phase of the project was started where number example simulations were simulated under different scenarios on *PopSim*. These examples were also simulated on *Plabsoft* so that a comparison between the output and speed of execution between the two could be made. It was observed that as far as the results of the simulations were concerned *PopSim* was producing results on par with those of *Plabsoft* irrespective of which simulation was compared. Though the major part of testing was performed on the Linux machine, some of these examples were also simulated on the Windows system to check whether the outcome of *PopSim* running on the Windows machine is similar to or different than *PopSim* running on Linux machine. It was observed that the output of *PopSim* on both of the systems was similar.

In the speed comparisons of performing the artificial example it was observed that *PopSim* was faster than *Plabsoft*, and the difference in execution time between the two

amplified on increasing the size of the population but diminished on increasing the number of repetitions. The reason for this variation was that on increasing the size of the population, a higher percentage of the simulation could be parallelized whereas on the other hand since the repetition loop was being controlled by the example script outside of *PopSim*, the percentage of the simulation that could be parallelized was smaller.

In case of the marker-assisted backcrossing examples, which considerably incorporated the parallelized part of *PopSim*, an average speed improvement of 53% over *Plabsoft* was observed with a maximum time saving of 60% and minimum time saving of 46%.

While in the case of the two QTL mapping examples the time advantage was negligible since these examples did not utilize the parallelized part of *PopSim*.

It was noted that while using *PopSim* an improvement of 75% (100 - 100/number of cores) could not achieved over *Plabsoft* on a four-core machine. The reasons for this were that these scripts also used some of the non-parallelized part of *PopSim* and the code of the script, which was outside of *PopSim*, is not multithreaded. Moreover, there was a little overhead caused by creating and managing treads. Apart from these reasons the interference of other processes running on the machine also affects the performance. For example, on a lightly loaded quad core machine if there was a very processor-heavy single threaded process the machine would give one core entirely to that process and would move all of the other processes to the remaining lightly-used three cores of the processor. On the other hand, the multithreaded process attempts to use all cores of the processor. Here the operating system does not have any vacant cores left to run its own processes and to which it can move the other processes. Therefore, in this case the operating system would schedule the threads so that the other processes of the system including the processes of the operating system itself get some of the processor time.

One can clearly infer from the present discussion that by using the new created software (*PopSim*) instead of *Plabsoft*, a considerable improvement in performance is achievable and *PopSim* compiles cleanly on both the Windows and the Linux systems. In addition, all of the requirements of the project were achieved.

5. Summary

All computer manufacturers have started implementing multiple cores in their computer systems to increase performance without increasing the cost exponentially. The main hindrance in the way of fully utilizing the gain in performance achieved by using these multi-core computers is that the programs, which were originally designed for sequential execution on a single core machine, are therefore only capable of using just one of the available cores of the processor at a time. This nullifies the benefits provided by system employing multiple cores. The newer software has to be designed in such a way that they break the problem into discrete parts that can then be performed concurrently on multi-core systems.

In designing a software for multiple core architecture where various parts of the problem are handled in tandem, a completely new set of obstacles have to be surmounted, such as race conditions, mutual exclusion, synchronization and parallel slowdown. Since there are no currently existing algorithms which can handle these barriers in case of population genetics simulations, it was essential to specifically design and implement new algorithms to handle population genetics problems on multi-core computers.

The objective of this project was to study various approaches of parallelization and to find the most suitable approach for implementation of parallelism. It involved analyzing the working of already existing simulation software *Plabsoft* and *Plabsim* in the beginning and then designing the new parallelized simulation software based on *Plabsoft*.

The first step after studying the workings of the existing software involved developing and testing an intermediate version of *Plabsoft*. This intermediate version was not dependent upon those three external libraries that the original *Plabsoft* was dependent on. New algorithms were designed for this, using the algorithms from the *Numerical Recipes (Press, Teukolsky, Vetterling and Flannery 2007)* as reference so that externally they mimicked the functioning of the random numbers and Poisson distribution generator found in the *GSL* library.

The directives-based parallel programming model was utilized for design and implementation of the new software during the second step. *OpenMP* was selected for the implementation of the directives-based parallel programming model. The parts of the program where multithreading was possible were noted, e.g. *crossing*, *genotype population* and *population evaluation*. After this, new algorithms supporting multithreading were designed to replace the old ones and coding of the new software *PopSim* was initiated.

After completion of the development phase of the project, the accuracy and reliability of the results of simulations performed by the newly created software *PopSim* along with the speed of execution of the simulations were tested on a four-core system. The testing was done using one artificial example simulation, eight marker-assisted backcrossing examples originally used in '*Selection strategies for marker-assisted backcrossing with high-throughput marker system (Herzog and Frisch 2011)*' and two QTL mapping examples originally used in '*A comparison of tests for QTL mapping with introgression libraries containing overlapping and non-overlapping donor segments (Mahone et al. 2012)'.* The artificial example simulated the offspring in Mendel's experiment where several differentiating characters were associated by simultaneously taking three characters i.e. form of seed, color of albumen and color of seed coat.

The example simulations were performed on both *PopSim* and *Plabsoft* under different settings on the Linux system. Moreover, some of the examples were additionally run by *PopSim* on the Windows system to confirm that the results from Windows and Linux system are equivalent.

After running the new software *PopSim* through these tests, it was observed that *PopSim* produced results, which were identical to the results produced by *Plabsoft* for all the examples. In case of the artificial example, it was noted that on increasing the size of the population the time difference between the execution of the simulation by *PopSim* and *Plabsoft* also increased. This was due to the fact that by increasing the population size, the percentage of the simulation which can be parallelized also increased. Conversely, with an increase in the number of repetitions in Artificial Example, the time difference decreased

because in this case *PopSim* could only parallelize one repetition at a time, as the loop running the repetitions was a sequential piece of code run by the parent script outside of *PopSim*. Significant improvement of speed was observed for the eight marker-assisted backcrossing examples as these examples engaged the parallelized components of *PopSim*. In case of the marker-assisted backcrossing examples, *PopSim* was able to give an average time savings of 53% with highest time savings of 60% and the lowest time savings of 46% over *Plabsoft*. However, the improvement in time was insignificant in case of the QTL mapping examples, as these examples did not utilize the parallelized part of *PopSim*. Moreover, the load time of *PopSim* was only 1/3 of the *Plabsoft*'s load time. In addition, it was also established that *PopSim* compiles cleanly under both Linux and Windows operating systems without giving any errors or warnings.

The new software *PopSim* is capable of utilizing the advantage provided by the multiple core architecture of modern computers while being completely backwards compatible with *Plabsoft* commands. Therefore, all scripts originally written for *Plabsoft* can be executed on *PopSim* without any modification. *PopSim* is designed to compile and run on both Linux and Windows operating systems and unlike *Plabsoft* it is not dependent on third party external libraries like GSL, GMP and others. Furthermore, the new software is also capable of being compiled on older compilers that do not support parallelism.

It could be concluded after extensive testing and comparing that by utilizing the multi-core architecture of newer computers, *PopSim* demonstrates a considerable and significant improvement in performance over *Plabsoft*.

6. References

Architecture Review Board 1997. *OpenMP*. http://openmp.org/wp/

Balloux F. 2001. *EASYPOP (Version 1.7): a computer program for population genetics simulations*. Journal of Heredity 92:301-302.

Becker R., Chambers J. 1981. *S: a language and system for data analysis. Bell laboratories computer information service*. Murray Hill, NJ, USA.

Bulmer M. 1985. *The mathematical theory of quantitative genetics*. Oxford University Press, USA, pp: 252.

Elliston B., Johnson R. 1998. *Pthreads-w32*. http://sourceware.org/pthreads-win32/

Excoffier L., Laval G., Schneider S. 2005. *Arlequin (Version 3.0): an integrated software package for population genetics data analysis*. Evolutionary Bioinformatics 1: 47–50.

Frisch M., Bohn M., Melchinger A.E. 2000. *Plabsim: software for simulation of marker-assisted backcrossing*. Journal of Heredity 91: 86–87.

Garnier-Gere P., Dillmann C. 1992. *A computer program for testing pairwise linkage disequilibria in subdivided populations*. Journal of Heredity 83: 239.

Gropp W., Lusk E., Skjellum A. 1994. *Using MPI*: *portable parallel programming with the message-passing interface*. MIT Press Scientific and Engineering Computation Series, Cambridge, MA, USA, pp: 307.

Haldane J. 1919. *The combination of linkage values and the calculation of distances between linked factors*. Journal of Genetics 8: 299–309.

Herzog E., Frisch M. 2011. *Selection strategies for marker-assisted backcrossing with high-throughput marker systems*. Theoretical and Applied Genetics 123: 251-260.

IEEE Std 1003.1c-1995. *POSIX Threads*.

Ihaka R., Gentleman R. 1996. *R: a language for data analysis and graphics*. Journal of Computational and Graphical Statistics 5: 299–314.

Karlin S., Liberman U. 1978. *Classification of multilocus recombination distributions*. Proceedings of National Academy of Sciences USA. 75: 6332–6336.

Kernighan B.W., Ritchie D.M. 1988. *The C programming language, (2nd edn)*, Prentice Hall, Englewood Cliffs, NJ, USA, pp: 274.

Laval G., Excoffier L. 2004. *SimCoal 2: a program to simulate genomic diversity over large recombining regions in a subdivided population with a complex history*. Bioinformatics 20: 2485-2487.

Mahone G., Borchardt D., Presterl T., Frisch M. 2012 *A comparison of tests for QTL mapping with introgression libraries containing overlapping and non-overlapping donor segments*. Crop Science (In Press)

Maurer H.P., Melchinger A.E., Frisch M. 2008. *Population genetic simulation and data analysis with Plabsoft*. Euphytica 161: 133–139.

O'Fallon B. 2010. *TreesimJ: a flexible, forward time population genetic simulator*. Bioinformatics 26: 2200-2201.

Peng B., Kimmel M. 2005. *simuPOP: a forward-time population genetics simulation environment*. Bioinformatics 21: 3686-3687.

Podlich D.W., Cooper M. 1998. QU-GENE: a *platform for quantitative analysis of genetic models*. Bioinformatics 14: 632-653.

Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P. 2007. *Numerical recipes: The art of scientific computing, (3rd edn)*, Cambridge University Press, New York, USA, pp: 1234.

Prigge V., Maurer H.P., Mackill D.J., Melchinger A.E., Frisch M. 2008. *Comparison of the observed with the simulated distributions of the parental genome contribution in two marker assisted backcross programs in rice*. Theoretical and Applied Genetics 116: 739-744.

Raymond M., Rousset F. 1995. *GENEPOP (Version 1.2): population genetics software for exact tests and ecumenicism*. Journal of Heredity 86: 248–249.

Reinders J. 2007. *Intel threading building blocks: outfitting C++ for multi-core processor parallelism*. O'Reilly Media Inc., Sebastopol, CA, USA, pp: 336.

Slatkin M. 1993. *Isolation by distance in equilibrium and non-equilibrium populations*. Evolution 47: 264-279.

Spencer C.C., Coop G. 2004. *SelSim: a program to simulate population genetic data with natural selection and recombination*. Bioinformatics 20: 3673–3675.

Stallman R., Tower L. 1987. *GCC, the GNU Compiler Collection*. http://gcc.gnu.org/

Stam P. 1979. *Interference in genetic crossing over and chromosome mapping.* Genetics 92: 573–594.

Swofford D.L., Selander R.B. 1981. *Biosys-1: a FORTRAN program for the comprehensive analysis for electrophoretic data in population genetics and systematics*. Journal of Heredity 72: 281-283.

Thompson K., Ritchie D., Kernighan B., McIlroy D., Ossanna J. 1969. *UNIX*. http://www.unix.org/

7.1 Appendix - I - List of PopSim Commands

Command	Description
be.quiet	The function *be.quiet()* filters the program messages, only warnings and errors are printed on the screen. Usage: be.quiet()
cross	Crosses between randomly or determinately chosen individuals of one or two parent populations are made. If *self* and *mode* is not declared then no selfing and self-compatibility is assumed. If *NoPg* is not declared then an already existing population, *NamePg* is refilled with new generated individuals. Usage: cross(NamePg, NameP1, NameP2, NoPg=-1, self=0, mode=1, crossing.scheme=0) Arguments: *NamePg* Name of the progeny. *NameP1* Name of the 1st parent population (Population A) *NameP2* Name of the 2nd parent population (Population B) *NoPg* Number of Individuals to be generated *self* Selfing rate [0-1]; self = 0.5 partial selfing; self = 0: no selfing (default); self =1: complete selfing *mode* Self-incompatibility [true mode=0| false mode=1] *crossing.scheme* crossing scheme (0-3)
define.effects	This function assigns effects to alleles at loci and to combinations of an arbitrary number of alleles, which may be located at different loci. The specifications are stored in text files. These can be generated either with an editor or with R functions. Here the effect file is automatically generated, loaded and the text file, containing the specification, is removed. Usage: define.effects(effectfile,description) Arguments: *effectfile* A string containing the name of the effect file and the name of the effect itself *description* A string containing a description of the effects to be *generated crossing.scheme* crossing scheme (0-3)

define.map	The function *define.map()* loads a linkage map into memory Here the linkage map file is automatically generated, loaded and the text file, containing the specification, is removed. Usage: define.map(description) Arguments: *description* A string containing a description of the linkage map
dh	The function *dh* produces one double haploid progeny of randomly chosen individuals of the initial population *NameP1*. All produced individuals are put together in one population *NamePg*. If *NoPg* is not declared then an already existing population, *NamePg* is refilled with new generated individuals. Usage: dh(NamePg, NameP1, NoPg) Arguments: *NamePg* Name of the population to be produced *NameP1* Name of the original population *NoPg* Number of individuals to be generated
evaluate.allele	The function *evaluate.allele* returns a statistic about the distribution of the alleles at specified marker loci. This function counts the occurrence of each allele at the specified *loci*. Usage: evaluate.allele(PopName,eLoci) Arguments: *PopName* Name of the population to be evaluated *eLoci* Names of the loci to be evaluated
evaluate.genome	The function *evaluate.genome()* computes the frequency distribution of an allele across the whole genome. Usage: evaluate.genome(PopName,all) Arguments: *PopName* Name of the population to be evaluated *all* Allele of which the frequency is counted

evaluate.genotype	The function *evaluate.genotype* prints a statistic about the distribution of the alleles. Usage: evaluate.genotype(PopName, eLoci, alleles=0, mode=2) Arguments: *PopName* Name of the population to be evaluated *eLoci* A List of Loci *alleles* A list of different Alleles *mode*
evaluate.mdp	The function *evaluate.mdp* calculates the number of marker data points required in a marker assisted selection program for a specified cross. This function considers only the marker listed in the effectfile. Usage: evaluate.mdp(NamePg, NameP1, NameP2, effectfile) Arguments: *NamePg* Name of population from which is selected *NameP1* Name of the first parent population *NameP2* Name of the second parent population *effectfile* Name of the effect to evaluate mdp
evaluate.population	The function *evaluate.population()* computes for every individual of the populations listed in the string *PopNames* its selection index. Usage: evaluate.population(PopNames,effName=NULL) Arguments: PopNames Name of the populations to be evaluated effName Name of the effect file
generate.effect.file	The function *generate.effect.file()* generates simple effect files automatically, as an alternative to specify an effect file by writing it down with an editor. Only additive and dominance effects can be generated automatically. Usage: generate.effect.file(fName,description) Arguments: *fName* A string containing the name of the effect file *description* A string containing a description of the effects to be generated

generate.map.file	The function *generate.map.file* generates a map file automatically - instead of specifying a linkage map by writing it down with an editor. Usage: generate.map.file(fName,description) Arguments: fName A string containing the name of the map file description A string containing a description of the linkage map
generate.population	The function *generate.population()* creates new populations from a matrix. Usage: generate.population(dta) Arguments: dta Data frame with a population description
genome.contribution	The function *genome.contribution()* makes statistics for the distribution of an *allele* within a list of populations. Usage: genome.contribution(pops,allele) Arguments: *pops* Names of the populations separated by blanks *allele* Allele which is considered
get.genome.par	The function *get.genome.par()* Usage: get.genome.par()
genome.parameter.set	The function *genome.parameter.set()* Usage: genome.parameter.set(no.chrom,no.hom,chrom.len) Arguments: *no.chrom* Number of chromosomes *no.hom* Number of homologues. *chrom.len* A list of the chromosome lengths
genotype.population	The function *genotype.population()* calculates the allelic composition of all loci defined in a previous loaded linkage map. Usage: genotype.population(PopNames) Arguments: *PopNames* Name of the populations to be genotyped
get.map	The function get.map()

get.mdp	The function *get.mdp()* returns the number of required marker data points till now. Usage: get.mdp()
get.population.gvalue	The function *get.population.gvalue()* returns a field with the calculated genotypic index for every individual of the population *pop*. If an effect is specified then the value of this effect is returned Usage: get.population.gvalue(name,EffName=NULL) Arguments: *name* Name of the population *EffName* Name of the effect
get.population	The function *get.population()* returns a population and creates a data frame from it. Usage: get.population(PopName) Arguments: *PopName* Name of the population
get.score	The function *get.score()* returns a field with the calculated genotypic index for every individual of the population pop. If an effect is specified then the value of this effect is returned. Usage: get.score(pop,effectfile=NULL) Arguments: *pop* Name of the evaluated population. *effectfile* Name of effect, which should be evaluated.
homozygote	The function *homozygote()* produces a data frame of a population with *NoInd* individuals which carry at all loci the same allele (*allele*). This data frame can be used to initialize a population. Usage: homozygote(allele,NoInd=1) Arguments: allele Allele which is carried at all loci NoInd Number of individuals to be generated

init.population	The function *init.population()* initializes a population with data in *PopSim*'s own format Usage: init.population(name,data,delete=1) Arguments: name Name of the population which shall be initialized data Data frame with data in a suitable format delete Removing already existing population with the name *name* before executing the command init.population() (delete = 1 (default) -> existing population will be removed; delete = 0 -> existing population will not be removed).
linkage.map.load	A linkage map is loaded from a text file. A map file consists of an arbitrary number of locus definitions. The locus definitions are ordered according to the chromosome number and map position Each locus is defined as Chromosome Position Locusname Classname ▪ Chromosome: The number of the chromosome on which the locus is located. ▪ Position: The map position of the locus on the chromosome. The distance from the telomere measured in Morgan units. ▪ Locusname: The name of the locus. ▪ Classname: A class of Loci to which the locus belongs. ▪ [,. . .] An arbitrary number of loci can be defined. . Usage: linkage.map.load(fName) Arguments: *fName* Name of the input file
linkage.map.remove	A linkage map once loaded, can be removed in order to load an alternative map. Usage: linkage.map.remove()
list.effects	The function *list.effects()* shows all effects already loaded with their weights. Usage: list.effects()

load.effmap	The function *load.effmap()* assigns effects to alleles at loci and to combinations of an arbitrary number of alleles, which may be located at different loci. The specifications are stored in text files. These can either be generated with an editor or with R functions (define.effects() generate.effect.file()). Structure of an effect file: A map file consists of a value, which is assigned to each individual of a population (the population mean) and a list of effects which are added to the population mean if the certain allelic combinations occur in an individual. The order of the effects is arbitrary, but the population mean must occur first in the map file. ▪ Mean: The population mean ▪ Classname: A class of Loci to which effects are assigned to ▪ Allele: The alleles to which the effects are assigned ▪ [Allele]: Optional. Is used to define dominance effects or even more complex effects. ▪ [,. . .]: An arbitrary number of effects can be defined separated by commas Usage: load.effmap(fname) Arguments: *fName* Name of the input file
population.append	The *population.append()* function concatenates a copy of population *NameP2* to population *NameP1*. The copy of population *NameP2* is inserted into population *NameP1* at the end. The population *NameP2* is untouched by the operation. Usage: population.append(NameP1, NameP2) Arguments: *NameP1* Population which takes up the copied individuals *NameP2* The population to be appended
population.concat	The function *population.concat()* concatenates two populations. Population *NameP2* is appended at the end of population *NameP1*. In the end, population NameP2 is removed. Usage: population.concat(NameP1, NameP2) Arguments: *NameP1* Recipient population *NameP2* Name of the population to be concatenated

population.copy	The *population.copy()* function copies a sub-population of population *NameP2* into population *NameP1* starting with the individual at position start(The index of the first individual of population *NameP2* is 1). *n* is omitted, the sub-population will range to the end of population *NameP2*. If *n* is too large for the sub-population to fit in population *NameP2*, the result will be truncated at the end of population *NameP2*. Usage: population.copy(NameP1, NameP2, start=1, n=-1) Arguments: *NameP1* Name of the population to be created *NameP2* Name of the original population *start* Index of the first copied individual *n* Number of individuals to be copied
population.divide	The *population.divide()* function divides a population *NameP2* into two sections. The first *NoI* individuals are stored in population *NameP1*. The remainder remains in population *NameP2*. Usage: population.divide(NameP1, NameP2, NoI) Arguments: *NameP1* Name of the population to be created *NameP2* Name of the population to be divided *NoI* Number of individuals to be separated from population NameP2
population.list	The function *population.list* shows the names of all loaded populations. Usage: population.list()
swap.population.name	The function *swap.population.name()* swaps the names of two populations. Usage: swap.population.name(NameP1,NameP2) Arguments: *NameP1* Name of the 1st population *NameP2* Name of the 2nd population
optimize.population	The function *optimize.population()* optimizes the memory structure of population *PopName* by freeing unused memory. Usage: optimize.population(PopNames) Arguments: *PopNames* A string containing the names of the populations to be optimized

population.remove.all	All populations are deleted. Usage: population.remove.all()
population.remove	The *population.remove()* function erases a set of populations from the list of populations specified by *PopName* and frees the memory. The populations to be deleted are listed in a string specified by *PopName* separated by blanks. Usage: population.remove(PopNames) Arguments: *PopNames* Names of the populations to be deleted
rename.population	The function *rename.population()* renames a population. Usage: rename.population(OldName, NewName) Arguments: *OldName* Name of the population to be renamed *NewName* New name of the population
population.resize	The function *population.resize()* changes the population size into the new size *newSize*. If the new population size is smaller than the old size then the individuals at the end are deleted (If the population is sorted then the individuals with the smallest value are deleted). In the opposite case dummy individuals are added to the population. Usage: population.resize(PopName, newSize) Arguments: *PopName* Name of the population to be resized *newSize* The new population size
population.sample	The function *population.sample()* takes a random sample from population *NameP2*. The sample forms a new population NameP1 Usage: population.sample(NameP1, NameP2, size=-1, replace=FALSE) Arguments: *NameP1* Name of the sampled population *NameP2* Name of the initial population *size* non-negative integer giving the number of individuals to choose. *replace* Should sampling be with replacement?

population.size.get	The function *get.populations.size()* returns a data frame with two columns *NoInds* and *NoIndsAlloc*. ▪ NoInds: Number of individuals in the memory ▪ NoIndsAlloc: Number of individuals allocated Usage: population.size.get(PopName) Arguments: *PopName* Name of the population
population.sort	The function population.sort() sorts populations according to the genotypical value. Usage: population.sort(PopName, decreasing=TRUE) Arguments: *PopName* Name of the populations to be sorted *decreasing* Logical. Should the sort be increasing or decreasing?
remove.effmaps	The function *remove.effmaps()* removes all effects previous loaded. Usage: remove.effmaps()
remove.evaluate.population	The function *remove.evaluate.population()* frees the memory, allocated by the function *evaluate.population*. Usage: remove.evaluate.population(PopNames) Arguments: *PopNames* Name of the population
remove.genotype.population	The function *remove.genotype.population()* frees the memory, allocated by the function the *genotype.population* has allocated. Usage: remove.genotype.population(PopNames) Arguments: *PopNames* Names of the populations where the genotypes should be removed
reset.all	Resets the program into the initial state. Usage: reset.all()

reset.mdp	The function *reset.mdp()* resets the number of marker data points, which were required during the breeding program. The count of marker data points is a global variable. Usage: reset.mdp()
resources	List effect. Usage: resources(expr) Arguments: *expr* Name of Population 1
return.population.disk	The function *return.population.disk()* returns a data frame in the NTSYs-format containing several populations and their marker data. Usage: return.population.disk(PopNames,file) Arguments: *PopNames* A string contaning a list of population names separated by blanks. *file* The name of the file where the data will be saved.
return.population	The function *return.population()* returns a data frame in the NTSYs-format containing several populations and their marker data. Usage: return.population(PopNames) Arguments: *PopNames* A string contaning a list of population names separated by blanks.
select.all.best.intern	List effect Usage: select.all.best.intern(NamePop,n=1) Arguments: *NamePop* Name of the population to be selected *n* Number of classes

select.all.best	The function *select.n.best()* selects individuals belonging to the '*x*' superior classes of the population *oldPop*. Usage: select.all.best(newPop,oldPop,effectfile=NULL,x=1) Arguments: *newPop* Name of population where the selected fraction should be stored *oldPop* Name of the population for which the selection should be carried out *effectfile* Optional \— The selection is carried out only for this effect. *x* Number of superior classes selected.
select.n.best	The function *select.n.best()* selects a fixed number of the superior '*n*' individuals of the population *oldPop*. Usage: select.n.best(newPop,oldPop,effectfile=NULL,n=1) Arguments: *newPop* Name of population where the selected fraction should be stored *oldPop* Name of the population for which the selection should be carried out *effectfile* Optional \— the selection is carried out for this effect. *n* Number of individuals selected
set.co.freq	The function *set.co.freq()* changes the expected value for the crossover per Morgan. Usage: set.co.freq(cofreq=1) Arguments: *cofreq* expected number of crossovers per Morgan
set.eff.weight	The function *set.eff.weith()* sets the weight for a linear index calculated by *evaluate.population*. Usage: set.eff.weight(fname,weight) Arguments: *fname* Name of the effect *weight* Weight of the effect
set.info.level	The function *set.info.level()* sets the information level. The smaller the level number is the less program information is printed on the screen. Usage: set.info.level(level) Arguments: *level* information level

set.NoLociInit	The function *set.NoLociInit()* sets the number of alleles that are allocated at once. The default value is ten. Usage: set.NoLociInit(NoLociInit) Arguments: *NoLociInit* Number of loci allocated
set.popfile.path	The function set.popfile.path(). Usage: set.popfile.path(path) Arguments: *path* Path
ssd.mating	The function *ssd.mating()* produces from each individual of the initial population *NoPg* salved progeny. Usage: ssd.mating(NamePg,NameP,NoPg=1) Arguments: *NamePg* Name of the progeny *NameP* Name of the parent population *NoPg* Number of individuals to be generated
summarize.gvalue	The function *summarize.gvalue()*. Usage: ummarize.gvalue(pops,effectfile=NULL) Arguments: *pops* Names separated by blanks of the populations to be summarized.) *effectfile* Name of the effect
talk.to.me	The function talk.to.me() causes the program to print all messages on the screen. Usage: talk.to.me()
write.version	The function *write.version()* displays the revision number and the build date of the program library. Usage: write.version()

7.2 Appendix - II - Code of Artificial Example

```
library("PopSim")

######Start of Timer#####
timerZ <- Sys.time()
###################

options(width=120, digits=5)

# ----------------------
# Mendel's Experiments
# ----------------------
# --------------------
# Three characters
# --------------------

reset.all()
genome.parameter.set( no.chrom=3, no.hom=2,
chrom.len=c(0.01,0.01,0.01) )
define.map("fo trait 1/0, al trait 2/0, sc trait 3/0")
init.population("P1",homozygote(1))
init.population("P2",homozygote(8))
cross("F1","P1","P2",1)

cross("F2","F1","F1",1000)

evaluate.genotype("F2","fo al sc",c(8,8,8,8,8,8))

for (i in 1:100)
{
  cross("F2","F1","F1",1000)
  population.concat("Store","F2")
}

evaluate.genotype("Store","fo al sc",c(8,8,8,8,8,8))

######End of Timer#####
print(Sys.time()-timerZ)
###################
```

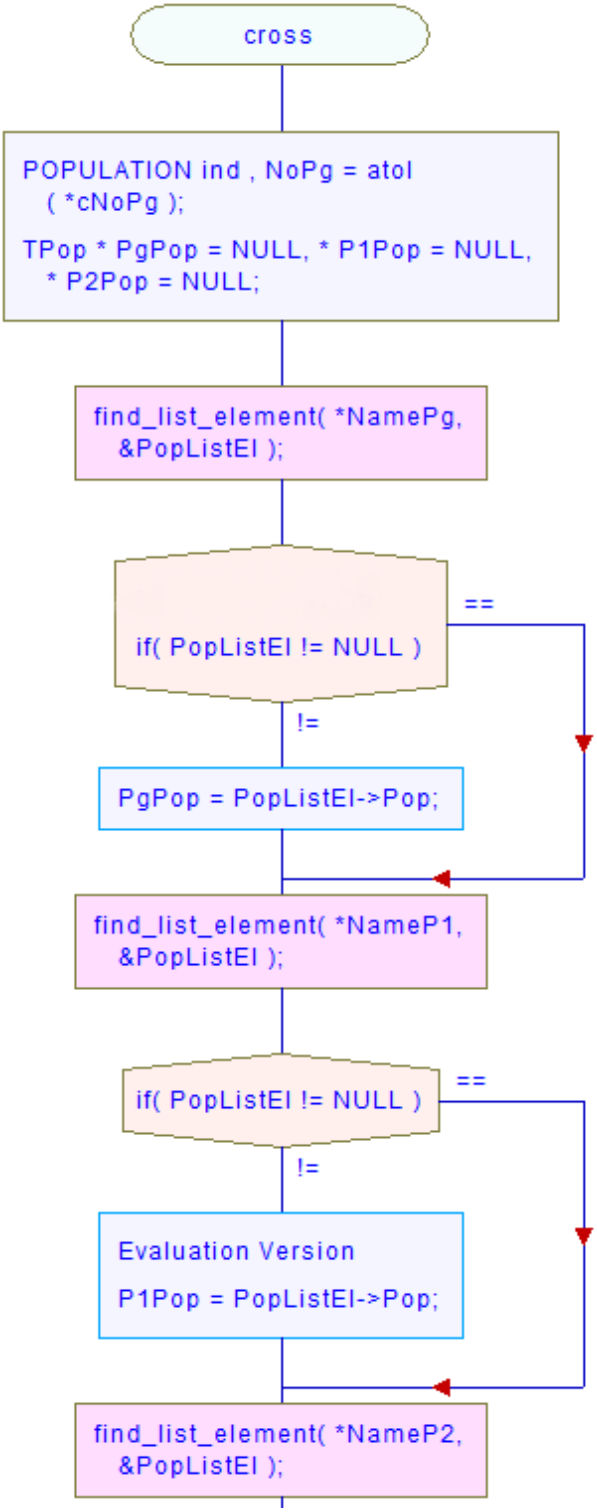
7.3 Appendix - III - 2cM Equally Example Script

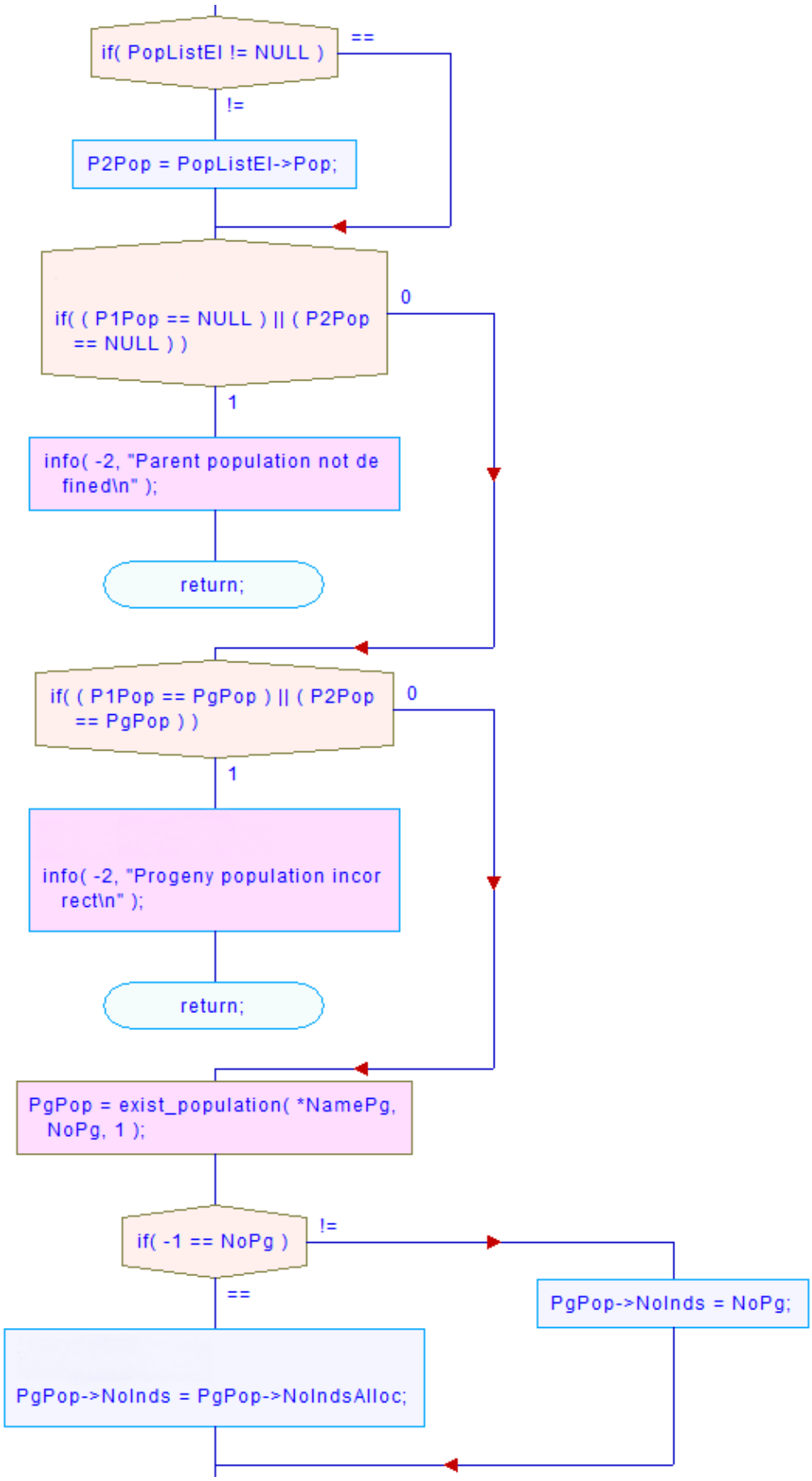
This script was originally used for the data in the Table 3 of '*Selection strategies for marker-assisted backcrossing with high-throughput marker system*' (*Herzog and Frisch 2011*).

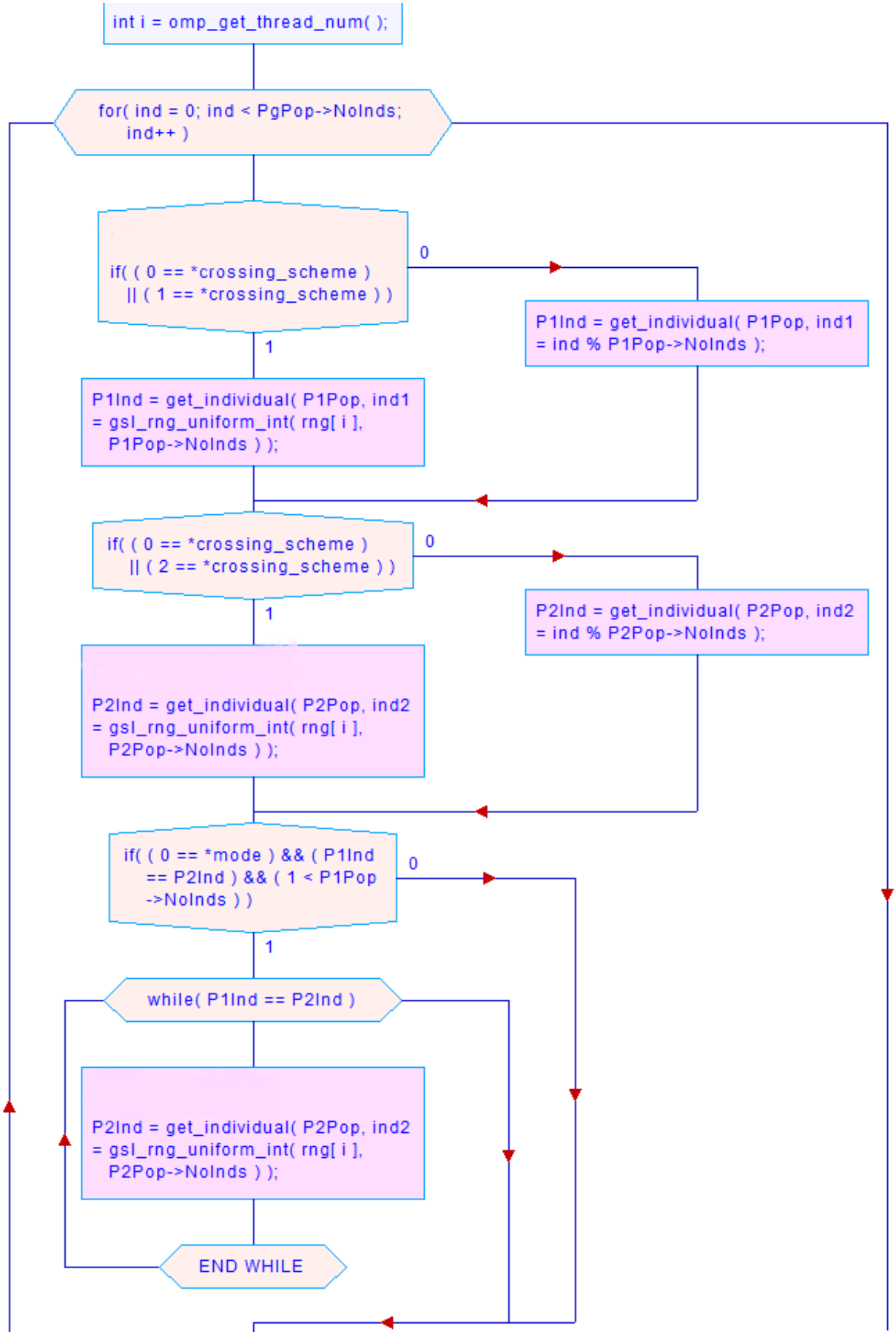
```
#################################################################
############
# Paper 1: High-Throughput

# Table 3: RPG recovered in BC 1-3 with two-stage selection

# Equally spaced markers

# Kartendichte 2 cM

# repetitions 10000
#################################################################
############

# equally spaced markers

library(PopSim)

set.genome.par(no.chrom=10,no.hom=2,chrom.len=rep(1.6,10))


load.linkage.map("zea-02-2.map")


define.effects("target", "0, target 1 uniform 1")
define.effects("markers","0, marker 8 uniform 1")

init.population("P1",homozygote(1))
init.population("P2",homozygote(8))

cross("F1","P1","P2",1)


pop <- c("F1","BC1","BC2","BC3")

s1     <- paste(pop,"s1",sep="")
s2     <- paste(pop,"s2",sep="")
sel    <- paste(pop,"sel",sep="")
st     <- paste(pop,"st",sep="")
sel[1] <- "F1"


popsize <- matrix(rep(seq(40, 200, 20), each = 3), ncol=3,
byrow=T)
```

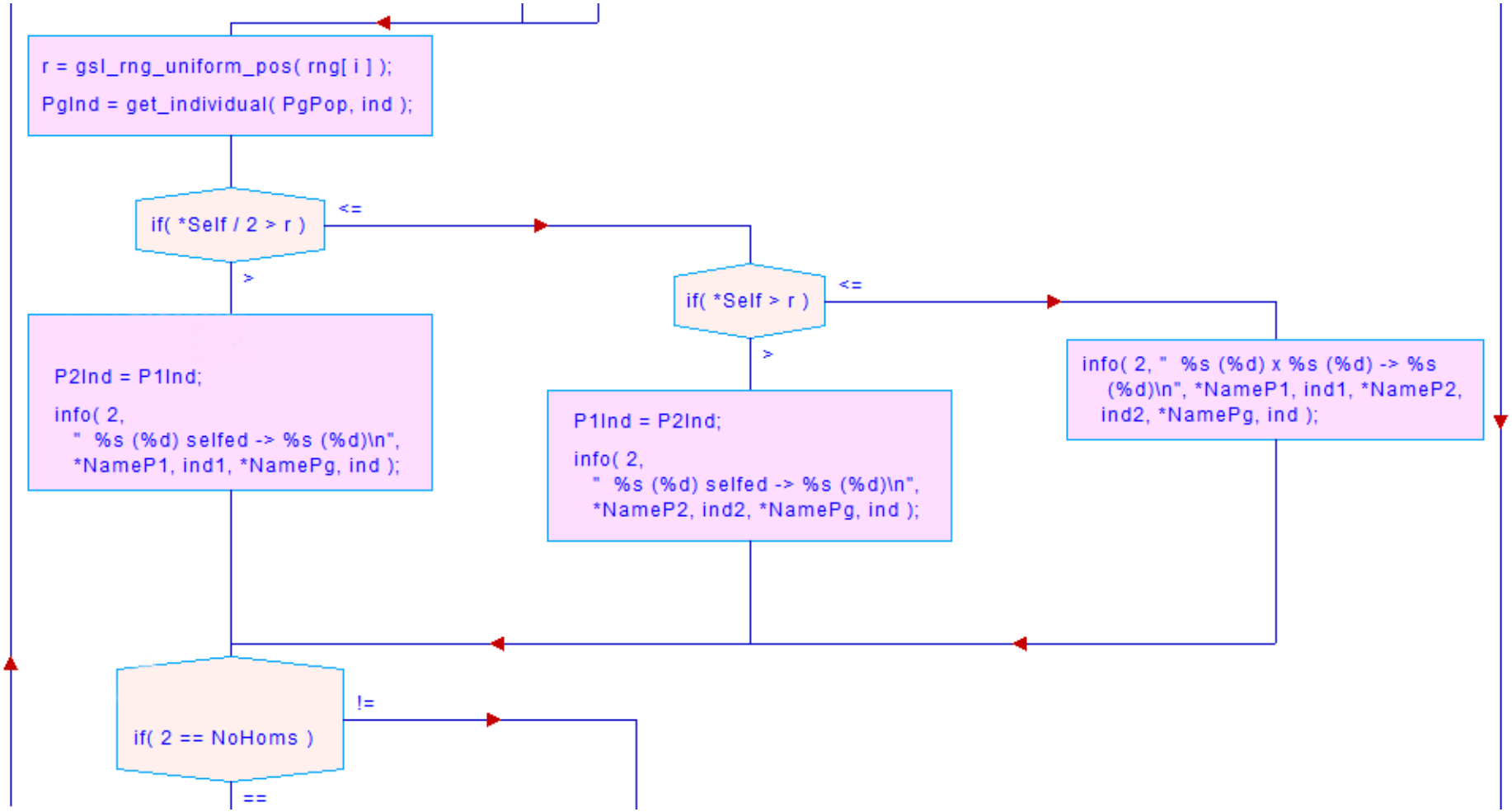
```
repetitions <- 10000

generations<-ncol(popsize)


Q10<-matrix(nrow=ncol(popsize),ncol=nrow(popsize))


m.d.points <-matrix(nrow=ncol(popsize),ncol=nrow(popsize))

{
for(k in 1:nrow(popsize)){
 n<-popsize[k,]
 mdp<-matrix(nrow=ncol(popsize),ncol=repetitions)
  for(j in 1:repetitions){
  reset.mdp()
   for(i in 1:generations){
   cross(pop[i+1],sel[i],"P2",n[i])
   select.all.best(s1[i+1],pop[i+1],"target")
   select.all.best(s2[i+1],s1[i+1],"markers")
   sample.population(sel[i+1],s2[i+1],1)
   append.population(st[i+1],sel[i+1])
  }
 }
 P2gen<-genome.contribution("BC1st BC2st BC3st",8)
 Q10[,k]<-P2gen[,5]
 remove.population(st)
}
}


RPG.Q10 <- data.frame( "n40"=Q10[,1], "n60"=Q10[,2],
"n80"=Q10[,3], "n100"=Q10[,4], "n120"=Q10[,5], "n140"=Q10[,6],
"n160"=Q10[,7], "n180"=Q10[,8], "n200"=Q10[,9])
rownames(RPG.Q10) <- c("BC1", "BC2", "BC3")
RPG.Q10

write.table(RPG.Q10, "P1-Table3-equally-2cM.dta")
```

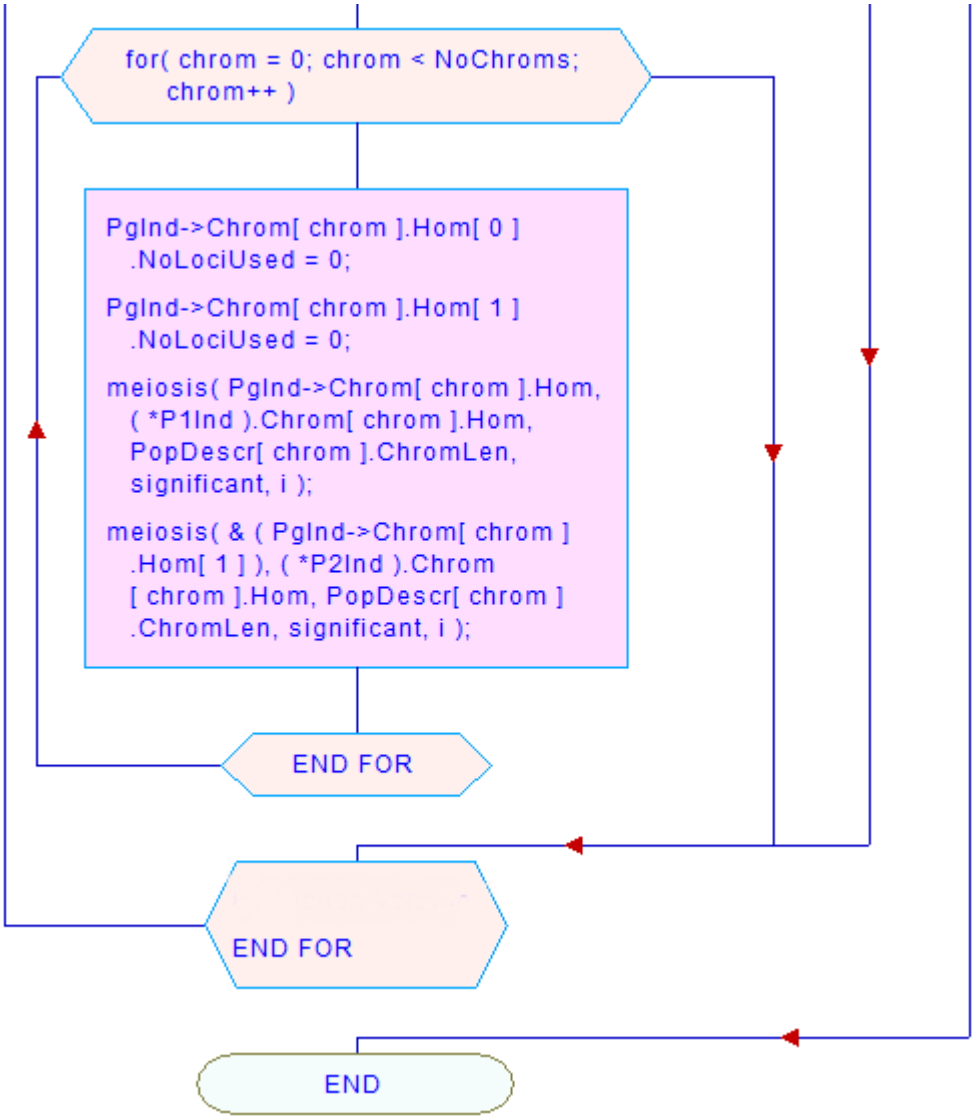
7.4 Appendix - IV - Flowchart of the Crossing Algorithm



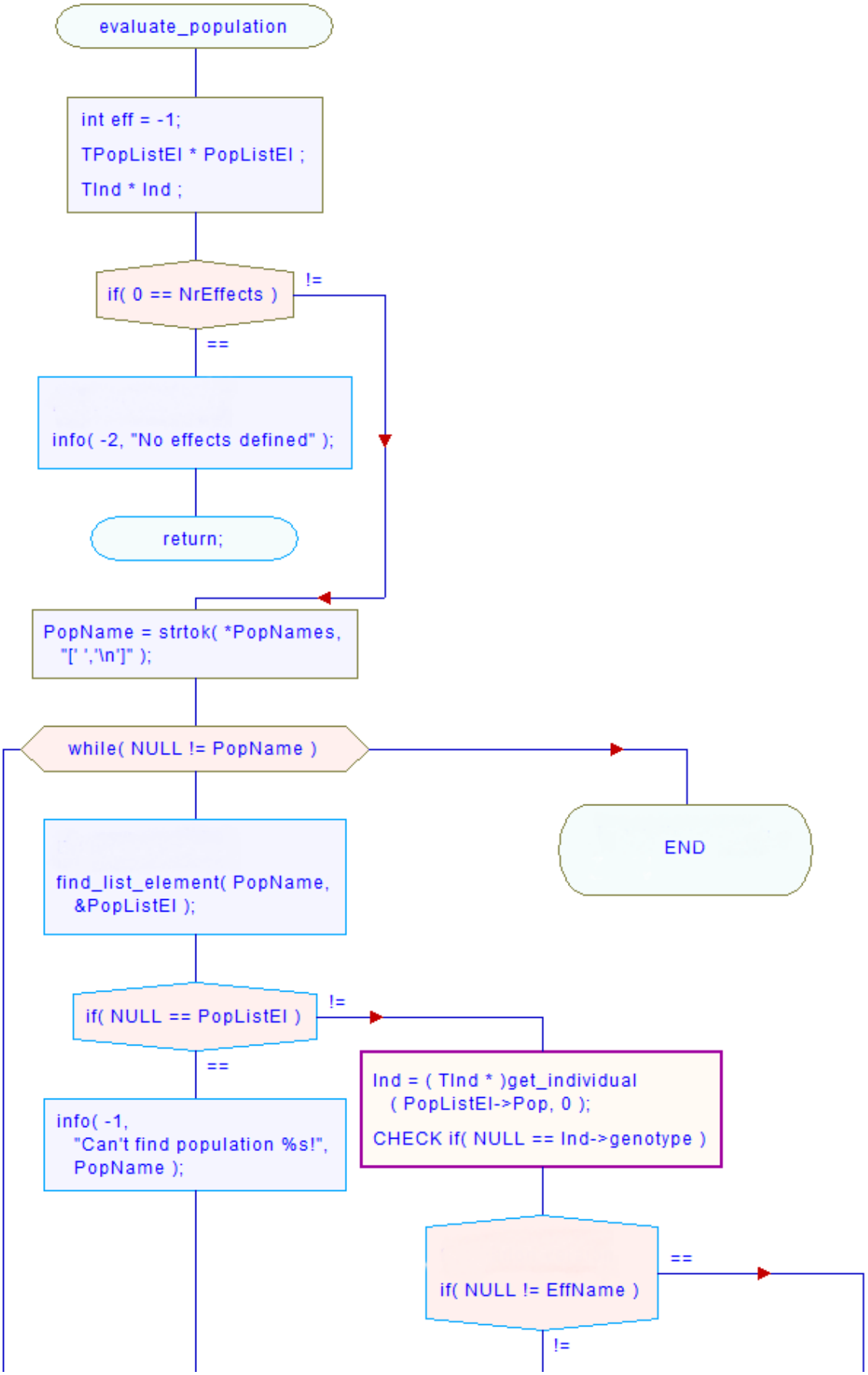

if(PopListEl != NULL)
==
!=
P2Pop = PopListEl->Pop;
if((P1Pop == NULL) || (P2Pop == NULL))
0
1
info(-2, "Parent population not de fined\n");
return;
if((P1Pop == PgPop) || (P2Pop == PgPop))
0
1
info(-2, "Progeny population incor rect\n");
return;
PgPop = exist_population(*NamePg, NoPg, 1);
if(-1 == NoPg)
!=
==
PgPop->NoInds = NoPg;
PgPop->NoInds = PgPop->NoIndsAlloc;

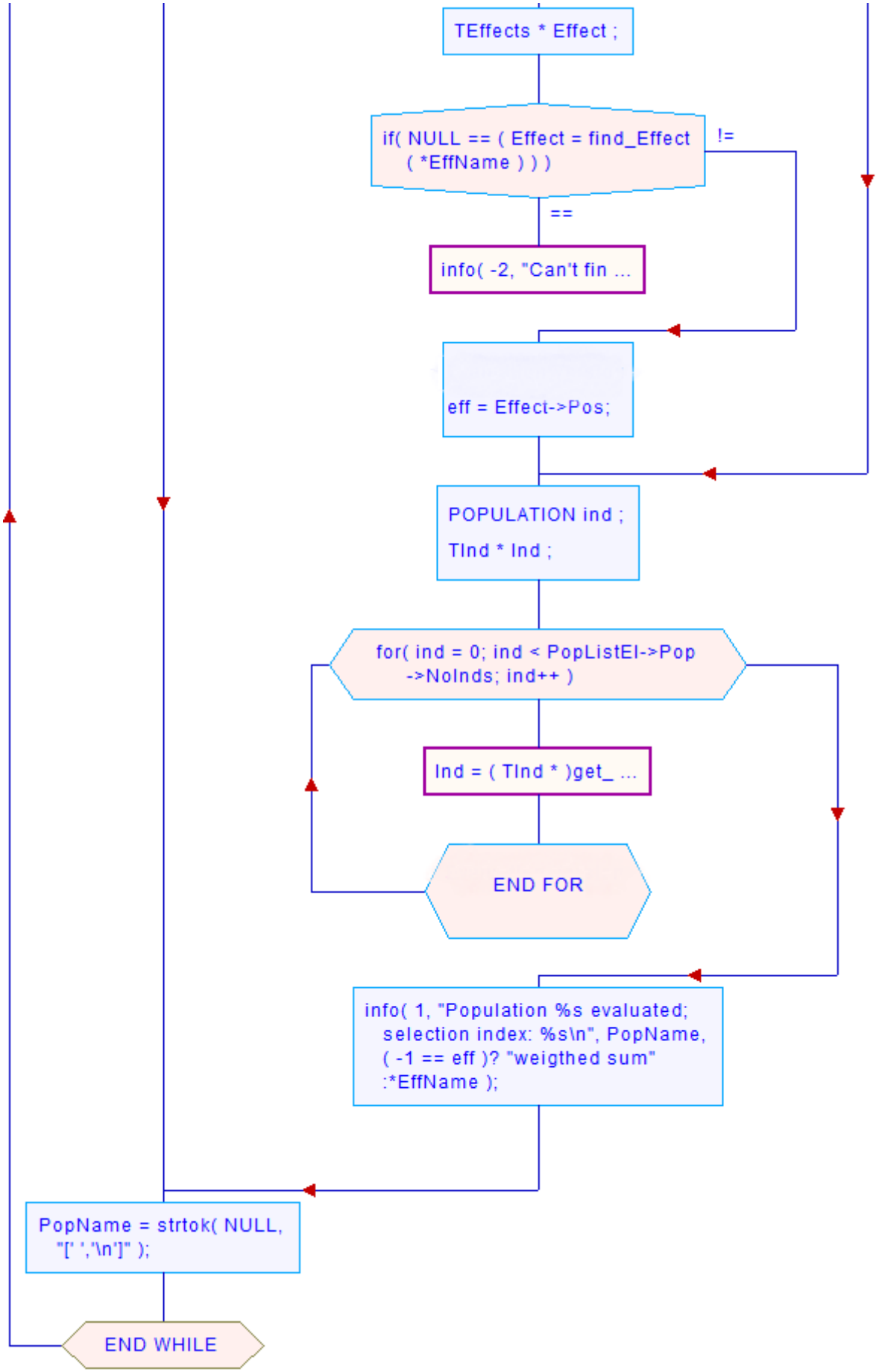

int i = omp_get_thread_num( );
for(ind = 0; ind < PgPop->NoInds; ind++)
if((0 == *crossing_scheme) || (1 == *crossing_scheme))
0
1
P1Ind = get_individual(P1Pop, ind1 = ind % P1Pop->NoInds);
P1Ind = get_individual(P1Pop, ind1 = gsl_rng_uniform_int(rng[i], P1Pop->NoInds));
if((0 == *crossing_scheme) || (2 == *crossing_scheme))
0
1
P2Ind = get_individual(P2Pop, ind2 = ind % P2Pop->NoInds);
P2Ind = get_individual(P2Pop, ind2 = gsl_rng_uniform_int(rng[i], P2Pop->NoInds));
if((0 == *mode) && (P1Ind == P2Ind) && (1 < P1Pop ->NoInds))
0
1
while(P1Ind == P2Ind)
P2Ind = get_individual(P2Pop, ind2 = gsl_rng_uniform_int(rng[i], P2Pop->NoInds));
END WHILE


r = gsl_rng_uniform_pos(rng[i]);
PgInd = get_individual(PgPop, ind);
if(*Self / 2 > r)
<=
>
P2Ind = P1Ind;
info(2,
" %s (%d) selfed -> %s (%d)\n",
*NameP1, ind1, *NamePg, ind);
if(*Self > r)
<=
>
P1Ind = P2Ind;
info(2,
" %s (%d) selfed -> %s (%d)\n",
*NameP2, ind2, *NamePg, ind);
info(2, " %s (%d) x %s (%d) -> %s
(%d)\n", *NameP1, ind1, *NameP2,
ind2, *NamePg, ind);
if(2 == NoHoms)
!=
==

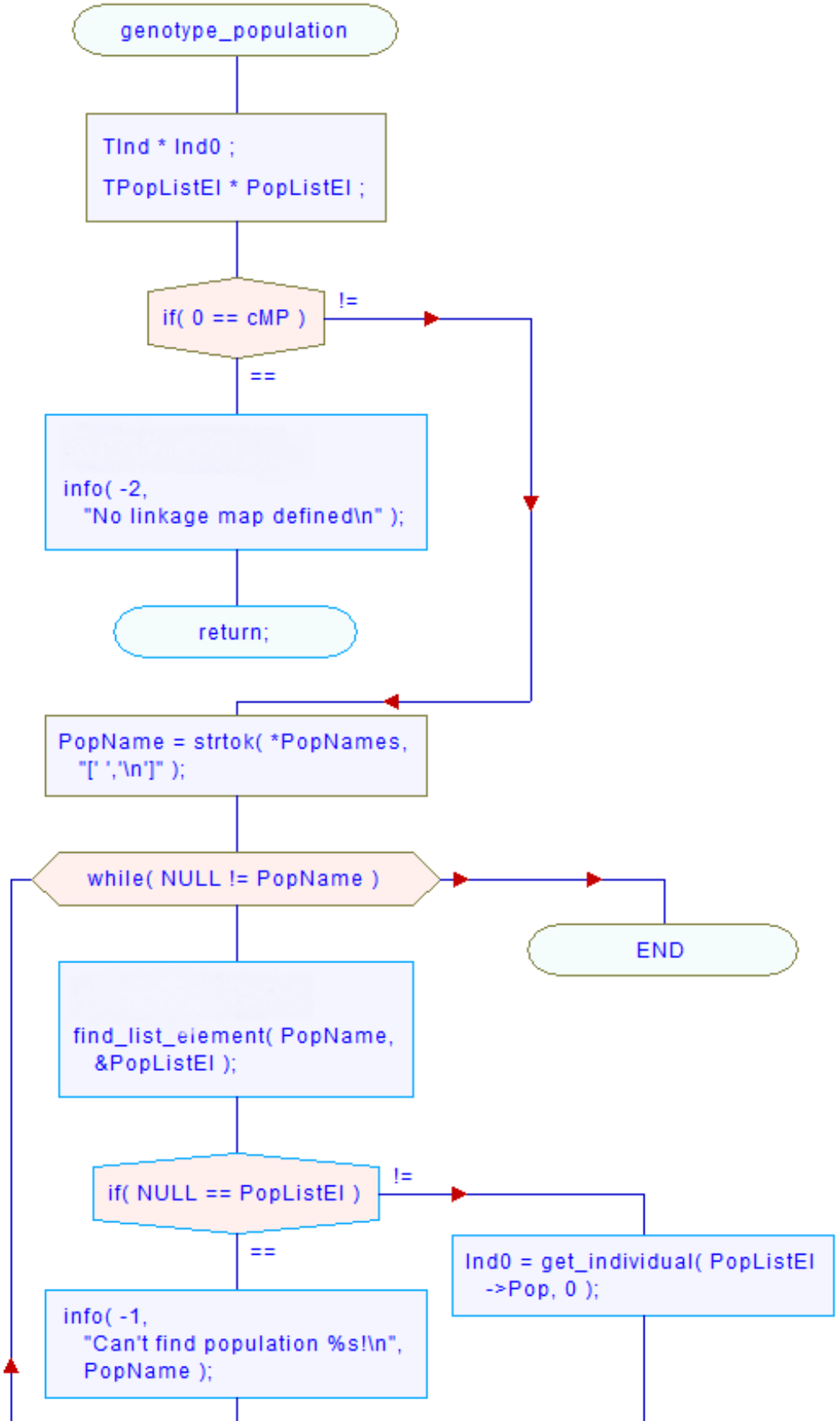

for(chrom = 0; chrom < NoChroms;
chrom++)
PgInd->Chrom[chrom].Hom[0]
.NoLociUsed = 0;
PgInd->Chrom[chrom].Hom[1]
.NoLociUsed = 0;
meiosis(PgInd->Chrom[chrom].Hom,
(*P1Ind).Chrom[chrom].Hom,
PopDescr[chrom].ChromLen,
significant, i);
meiosis(& (PgInd->Chrom[chrom]
.Hom[1]), (*P2Ind).Chrom
[chrom].Hom, PopDescr[chrom]
.ChromLen, significant, i);
END FOR
END FOR
END

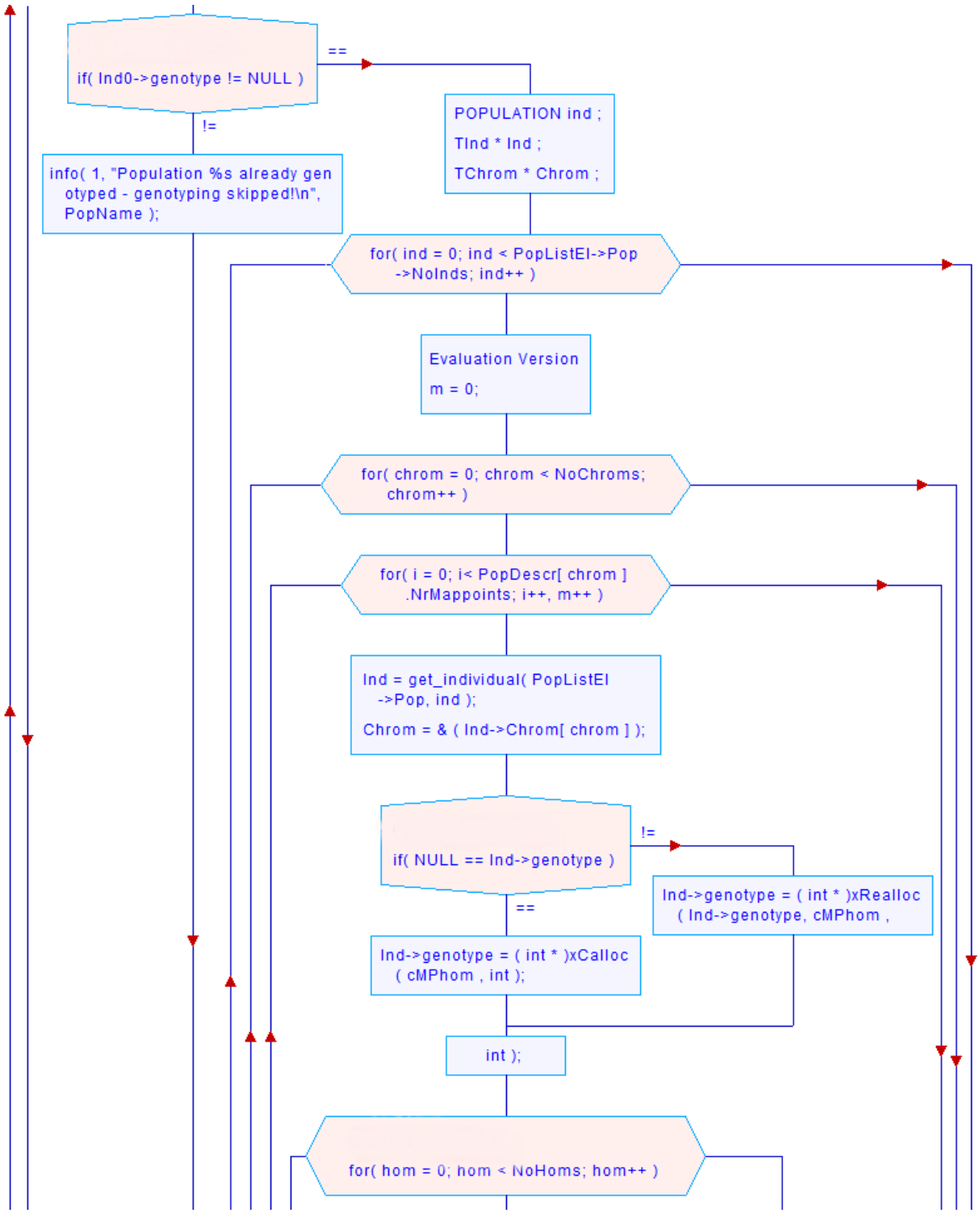
7.5 Appendix - V - Flowchart of the Population Evaluation Algorithm

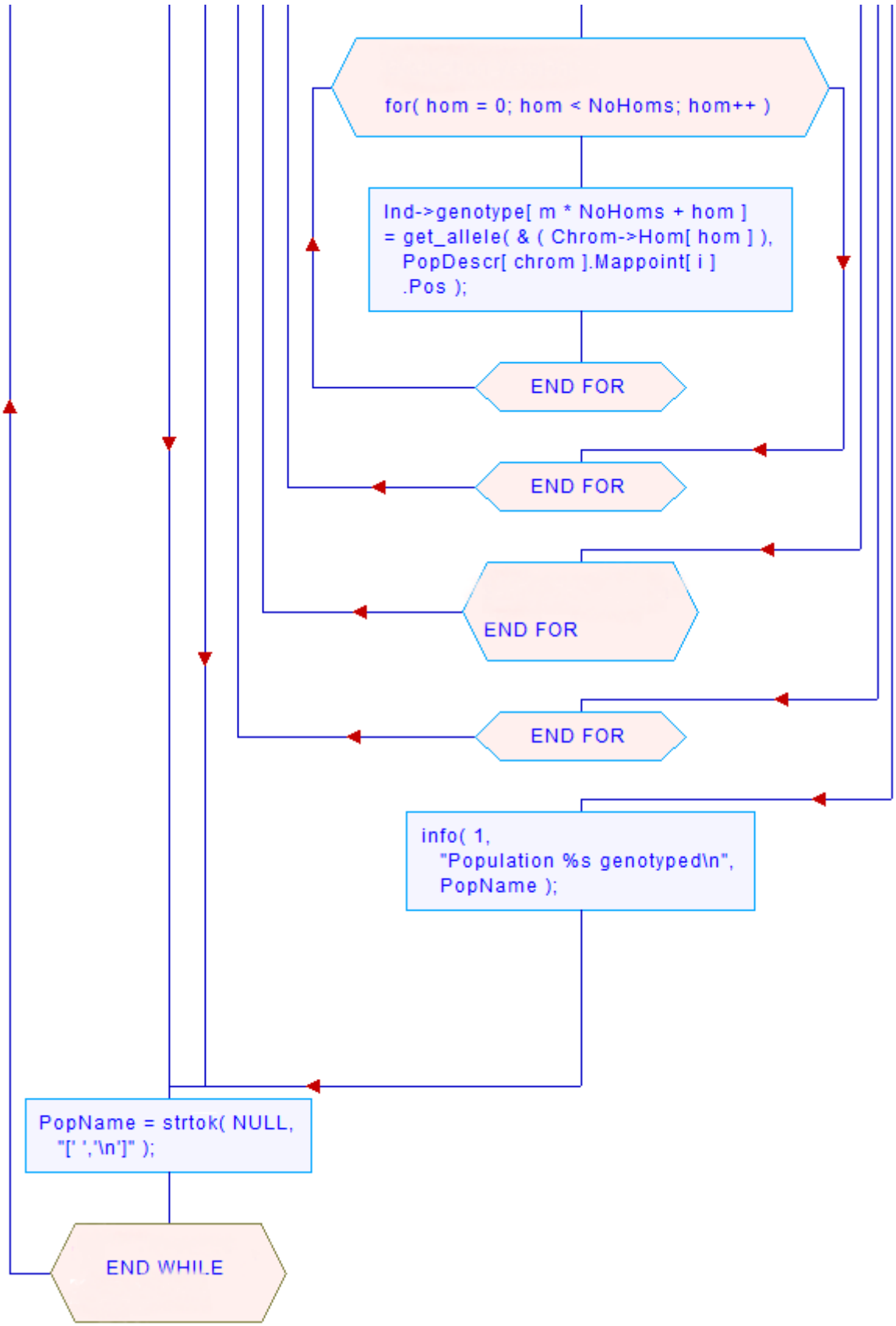



TEffects * Effect ;
if(NULL == (Effect = find_Effect (*EffName)))
!=
==
info(-2, "Can't fin ...
eff = Effect->Pos;
POPULATION ind ;
TInd * Ind ;
for(ind = 0; ind < PopListEl->Pop ->NoInds; ind++)
Ind = (TInd *)get_ ...
END FOR
info(1, "Population %s evaluated; selection index: %s\n", PopName, (-1 == eff)? "weigthed sum" :*EffName);
PopName = strtok(NULL, "[' ','\n']");
END WHILE

7.6 Appendix - VI - Flowchart of the Genotype Population Algorithm




if(Ind0->genotype != NULL)
==
!=
POPULATION ind ;
TInd * Ind ;
TChrom * Chrom ;
info(1, "Population %s already gen
otyped - genotyping skipped!\n",
PopName);
for(ind = 0; ind < PopListEl->Pop
->NoInds; ind++)
Evaluation Version
m = 0;
for(chrom = 0; chrom < NoChroms;
chrom++)
for(i = 0; i< PopDescr[chrom]
.NrMappoints; i++, m++)
Ind = get_individual(PopListEl
->Pop, ind);
Chrom = & (Ind->Chrom[chrom]);
if(NULL == Ind->genotype)
!=
==
Ind->genotype = (int *)xRealloc
(Ind->genotype, cMPhom ,
Ind->genotype = (int *)xCalloc
(cMPhom , int);
int);
for(hom = 0; hom < NoHoms; hom++)


for(hom = 0; hom < NoHoms; hom++)
Ind->genotype[m * NoHoms + hom]
= get_allele(& (Chrom->Hom[hom]),
PopDescr[chrom].Mappoint[i]
.Pos);
END FOR
END FOR
END FOR
END FOR
info(1,
"Population %s genotyped\n",
PopName);
PopName = strtok(NULL,
"[' ','\n']");
END WHILE

Declaration

Ich erkläre: Ich habe die vorgelegte Dissertation selbständig und ohne unerlaubte fremde Hilfe und nur mit den Hilfen angefertigt, die ich in der Dissertation angegeben habe. Alle Textstellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen sind, und alle Angaben, die auf mündlichen Auskünften beruhen, sind als solche kenntlich gemacht. Bei den von mir durchgeführten und in der Dissertation erwähnten Untersuchungen habe ich die Grundsätze guter wissenschaftlicher Praxis, wie sie in der "Satzung der Justus-Liebig-Universität Gießen zur Sicherung guter wissenschaftlicher Praxis" niedergelegt sind, eingehalten.

Acknowledgements

Acquiring of a Ph.D. is certainly a major milestone in the life of a person. As such, the present project was a challenge for me and presented an opportunity to prove my caliber.

This is a humble effort to express my sincere gratitude towards all those who have guided and helped me on the way to the completion of my research work and writing of my thesis towards the fulfillment of the requirements of the Ph.D. degree to be awarded by the Justus Liebig University.

If Prof. Dr. Mathias Frisch had not enrolled me as a Ph.D. student under him at the Justus Liebig University, it would not have been possible for me to begin and then finish this project. It was purely because of his experience, knowledge and constant guidance that I was able to clear all the theoretical and technical hurdles during the development phases of this project work. I am indeed indebted to him for all that which he has done for me.

I would like to take this opportunity to thank Prof. Dr. Thomas Wilke for being co-guide for my Ph.D. research project.

It would not have been possible for me to complete the testing of my project if my fellow Ph.D. students Eva Herzog and Gregory Mahone had not lent me their scripts to test the proper working of the program developed by me, for which I am obliged to them.

At this occasion, I would also like to thank Dr. Birgit Samans for helping me with the official forms, enrolment process and for the occasional German to English translations and Dr. Nathalie Steiner for explaining me the procedure and requirements for the submission of this thesis.

I would like to thank all of the members of the Biometry group for the constant support and help that they provided me throughout my stay in this institute.

In the end, the Deutsche Forschungsgemeinschaft is gratefully acknowledged for funding this research.

Arsh Rup Singh